

Machine Learning

Proceedings of the Fifteenth
International Conference (ICML '98)

Edited by Jude Shavlik

MADISON, WISCONSIN
JULY 24-27, 1998

DEPARTMENT OF THE ARMY
Approved for Public Release
Distribution Unlimited

19980810 049

MORGAN KAUFMANN

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE July 1998	3. REPORT TYPE AND DATES COVERED Final, 7/10/98 - 9/30/98		
4. TITLE AND SUBTITLE Machine Learning: Proceedings of the Fifteenth International Conference		5. FUNDING NUMBERS N00014-98-1-0810		
6. AUTHOR(S) Jude Shavlik (editor)				
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Computer Sciences Department University of Wisconsin 1210 W. Dayton Street Madison, WI 53706		8. PERFORMING ORGANIZATION REPORT NUMBER 144-HD17		
9. SPONSORING / MONITORING AGENCY NAMES(S) AND ADDRESS(ES) Office of Naval Research Program Officer Michael O. Shneir Code 311, Ballston Tower One 800 N. Quincy Street Arlington, VA 22217-5660		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release. (Proceedings may be purchased from Morgan Kaufmann Publishers.)		12. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) Contains 66 technical articles on recent international advances in the artificial intelligence subfield of machine learning. <div data-bbox="576 1467 1034 1617" data-label="Image"> </div>				
14. SUBJECT TERMS Machine learning, neural networks, genetic algorithms, artificial intelligence			15. NUMBER OF PAGES 580	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

University of Wisconsin - Madison

Jude W. Shavlik
Professor
Phone: (608) 262-7784
Fax: (608) 262-9777
Email: shavlik@cs.wisc.edu
URL: <http://www.cs.wisc.edu/~shavlik/>

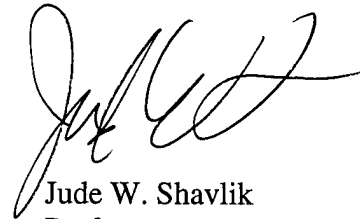
Department of Computer Sciences
1210 West Dayton Street
Madison, Wisconsin 53706
USA

August 5, 1998

Defense Technical Information Center
8725 John J. Kingman Road
STE 0944
Ft. Belvoir, VA 22060-6218

Please find enclosed one (1) copy of the proceedings of the Fifteenth International Conference on Machine Learning and one (1) copy of SF-298. Program Officer Michael O. Shneier of the Office of Naval Research provided partial financial support for this meeting.

Sincerely,



Jude W. Shavlik
Professor

MACHINE LEARNING

Proceedings
of the Fifteenth
International Conference
(ICML '98)

Edited by

Jude Shavlik
Department of Computer Sciences
University of Wisconsin, Madison

Madison, Wisconsin

July 24–27, 1998

DTIC QUALITY INSPECTED 1

MORGAN KAUFMANN PUBLISHERS
San Francisco, California

Production, design, type, and manufacturing management
provided by Professional Book Center, Denver, Colorado.

Cover image, satellite photograph of Madison, Wisconsin, courtesy
of the United States Geological Service.

Morgan Kaufmann Publishers, Inc.
Editorial Office:
340 Pine St., 6th Floor
San Francisco, CA 94104
<http://www.mkp.com>

ISBN 1-55860-556-8

ISSN 1049-1910

Copyright © 1998 by Morgan Kaufmann Publishers, Inc.
All rights reserved.

Printed in the United States

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or
by any means—electronic, mechanical, photocopying, recording, or otherwise without the prior written
permission of the publisher.

01 00 99 98 4 3 2 1

Contents

Preface

Advisory Committee

Program Committee

Tutorials

Workshops

Query Learning Strategies Using Boosting and Bagging	1
<i>Naoki Abe and Hiroshi Mamitsuka</i>	
Genetic Programming and Deductive-Inductive Learning: A Multi-Strategy Approach	10
<i>Ricardo Aler, Daniel Borrajo, and Pedro Isasi</i>	
An Experimental Evaluation of Coevolutionary Concept Learning	19
<i>Cosimo Anglano, Attilio Giordana, Giuseppe Lo Bello, and Lorenza Saitta</i>	
KnightCap: A Chess Program That Learns by Combining TD(λ) with Game-Tree Search	28
<i>Jonathan Baxter, Andrew Tridgell, and Lex Weaver</i>	
Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets	37
<i>Stephen D. Bay</i>	
Learning Collaborative Information Filters	46
<i>Daniel Billsus and Michael J. Pazzani</i>	
Top-Down Induction of Clustering Trees	55
<i>Hendrick Blockeel, Luc De Raedt, and Jan Ramon</i>	
A Supra-Classifer Architecture for Scalable Knowledge Reuse	64
<i>Kurt D. Bollacker and Joydeep Ghosh</i>	
Learning Sorting and Decision Trees with POMDPs	73
<i>Blai Bonet and Héctor Geffner</i>	
Feature Selection via Concave Minimization and Support Vector Machines	82
<i>Paul S. Bradley and Olvi L. Mangasarian</i>	
Refining Initial Points for K-Means Clustering	91
<i>Paul S. Bradley and Usama M. Fayyad</i>	
Finite-Time Regret Bounds for the Multiarmed Bandit Problem	100
<i>Nicolò Cesa-Bianchi and Paul Fischer</i>	
Bayesian Classifiers Are Large Margin Hyperplanes in a Hilbert Space	109
<i>Nello Cristianini, John Shawe-Taylor, and Peter Sykacek</i>	
The MAXQ Method for Hierarchical Reinforcement Learning	118
<i>Thomas G. Dietterich</i>	

A Process-Oriented Heuristic for Model Selection	127
<i>Pedro Domingos</i>	
Relational Reinforcement Learning	136
<i>Sašo Džeroski, Luc De Raedt, and Hendrik Blockeel</i>	
Generating Accurate Rule Sets Without Global Optimization	144
<i>Eibe Frank and Ian H. Witten</i>	
Using a Permutation Test for Attribute Selection in Decision Trees	152
<i>Eibe Frank and Ian H. Witten</i>	
Multistrategy Learning for Information Extraction	161
<i>Dayne Freitag</i>	
An Efficient Boosting Algorithm for Combining Preferences	170
<i>Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer</i>	
Bayesian Network Classification with Continuous Attributes: Getting the Best of Both Discretization and Parametric Fitting	179
<i>Nir Friedman, Moises Goldszmidt, and Thomas J. Lee</i>	
The Kernel-Adatron Algorithm: A Fast and Simple Learning Procedure for Support Vector Machines	188
<i>Thilo-Thomas Frieß, Nello Cristianini, and Colin Campbell</i>	
Multi-criteria Reinforcement Learning	197
<i>Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári</i>	
Local Cascade Generalization	206
<i>João Gama</i>	
A Learning Rate Analysis of Reinforcement Learning Algorithms in Finite-Horizon	215
<i>Frédéric Garcia and Seydina M. Ndiaye</i>	
Well-Behaved Borgs, Bolos, and Berserkers	224
<i>Diana F. Gordon</i>	
Solving a Huge Number of Similar Tasks: A Combination of Multi-Task Learning and a Hierarchical Bayesian Approach	233
<i>Tom Heskes</i>	
Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm	242
<i>Junling Hu and Michael P. Wellman</i>	
Coevolutionary Learning: A Case Study	251
<i>Hugues Juillé and Jordan B. Pollack</i>	
Near-Optimal Reinforcement Learning in Polynomial Time	260
<i>Michael Kearns and Satinder Singh</i>	
A Fast, Bottom-Up Decision Tree Pruning Algorithm with Near-Optimal Generalization	269
<i>Michael Kearns and Yishay Mansour</i>	
An Analysis of Actor/Critic Algorithms Using Eligibility Traces: Reinforcement Learning with Imperfect Value Function	278
<i>Hajime Kimura and Shigenobu Kobayashi</i>	

Using Learning for Approximation in Stochastic Processes	287
<i>Daphne Koller and Raya Fratkina</i>	
An Information-Theoretic Definition of Similarity	296
<i>Dekang Lin</i>	
Structural Machine Learning with Galois Lattice and Graphs	305
<i>Michael Liquiere and Jean Sallantin</i>	
Learning a Language-Independent Representation for Terms from a Partially Aligned Corpus	314
<i>Michael L. Littman, Fan Jiang, and Greg A. Keim</i>	
Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes	323
<i>John Loch and Satinder Singh</i>	
Learning to Locate an Object in 3D Space from a Sequence of Camera Images	332
<i>Dimitris Margaritis and Sebastian Thrun</i>	
Multiple-Instance Learning for Natural Scene Classification	341
<i>Oded Maron and Aparna Lakshmi Ratan</i>	
Employing EM and Pool-Based Active Learning for Text Classification	350
<i>Andrew Kachites McCallum and Kamal Nigam</i>	
Improving Text Classification by Shrinkage in a Hierarchy of Classes	359
<i>Andrew McCallum, Ronald Rosenfeld, Tom Mitchell, and Andrew Y. Ng</i>	
A Case Study in the Use of Theory Revision in Requirements Validation	368
<i>T. L. McCluskey and M. M. West</i>	
Stochastic Resonance with Adaptive Fuzzy Systems	377
<i>Sanya Mitaim and Bart Kosko</i>	
Q2: Memory-Based Active Learning for Optimizing Noisy Continuous Functions	386
<i>Andrew W. Moore, Jeff G. Schneider, Justin A. Boyan, and Mary S. Lee</i>	
Collaborative Filtering Using Weighted Majority Prediction Algorithms	395
<i>Atsuyoshi Nakamura and Naoki Abe</i>	
On Feature Selection: Learning with Exponentially Many Irrelevant Features as Training Examples	404
<i>Andrew Y. Ng</i>	
On the Power of Decision Lists	413
<i>Richard Nock and Pascal Jappy</i>	
An Analysis of Direct Reinforcement Learning in Non-Markovian Domains	421
<i>Mark D. Pendrith and Michael J. McGarity</i>	
A Randomized ANOVA Procedure for Comparing Performance Curves	430
<i>Justus H. Piater, Paul R. Cohen, Xiaoqin Zhang, and Michael Atighetchi</i>	
Classification Using Φ -Machines and Constructive Function Approximation	439
<i>Doina Precup and Paul E. Utgoff</i>	
The Case against Accuracy Estimation for Comparing Induction Algorithms	445
<i>Foster Provost, Tom Fawcett, and Ron Kohavi</i>	

Theory Refinement of Bayesian Networks with Hidden Variables	454
<i>Sowmya Ramachandran and Raymond J. Mooney</i>	
Learning to Drive a Bicycle Using Reinforcement Learning and Shaping	463
<i>Jette Randløv and Preben Alstrøm</i>	
Learning First-Order Acyclic Horn Programs from Entailment	472
<i>Chandra Reddy and Prasad Tadepalli</i>	
RL-TOPS: An Architecture for Modularity and Re-Use in Reinforcement Learning	481
<i>Malcolm R. K. Ryan and Mark D. Pendrith</i>	
Evolving Structured Programs with Hierarchical Instructions and Skip Nodes	488
<i>Rafał Szałustowicz and Jürgen Schmidhuber</i>	
An Investigation of Transformation-Based Learning in Discourse	497
<i>Ken Samuel, Sandra Carberry, and K. Vijay-Shanker</i>	
Automatic Segmentation of Continuous Trajectories with Invariance to Nonlinear Warpings of Time	506
<i>Lawrence K. Saul</i>	
Ridge Regression Learning Algorithm in Dual Variables	515
<i>C. Saunders, A. Gammerman, and V. Vovk</i>	
Value Function Based Production Scheduling	522
<i>Jeff G. Schneider, Justin A. Boyan, and Andrew W. Moore</i>	
Heading in the Right Direction	531
<i>Hagit Shatkay and Leslie P. Kaelbling</i>	
A Neural Network Model for Prognostic Prediction	540
<i>W. Nick Street</i>	
Learning the Grammar of Dance	547
<i>Joshua M. Stuart and Elizabeth Bradley</i>	
Intra-Option Learning about Temporally Abstract Actions	556
<i>Richard S. Sutton, Doina Precup, and Satinder Singh</i>	
Teaching an Agent to Test Students	565
<i>Gheorghe Tecuci and Harry Keeling</i>	
The Problem with Noise and Small Disjuncts	574
<i>Gary M. Weiss and Haym Hirsh</i>	
Author Index	579

Preface

This volume contains the 66 technical papers presented at the Fifteenth International Conference on Machine Learning (ICML '98), held July 24-27, 1998, in Madison, Wisconsin U.S.A. These articles were selected based on the rigorous review and discussion of 215 submissions. All papers were presented orally as well as at an evening poster session.

ICML '98 was one of ten AI-related conferences held in Madison during mid-summer 1998, in an ambitious, first-time experiment to see what kind of synergies would result from all this collocation. In particular, ICML '98 was held in the same building as, and concurrently with, the Computational Learning Theory (COLT) and Uncertainty in Artificial Intelligence (UAI) conferences. It also overlapped one day with the Inductive Logic Programming (ILP) conference.

Registrants were allowed to attend, without additional costs, the technical sessions of the other conferences. COLT, ICML, and UAI each invited one of the plenary speakers and jointly invited the banquet speaker, plus there was a joint poster session and wrapup panel. The poster session contained about 150 papers, allowing conference attendees a chance to see or further discuss research presented during the four to five parallel tracks and fostering interaction among the various communities.

I especially wish to thank:

- the authors of all the papers for their technical contributions toward the advancement of machine learning.
- Richard Sutton, who was the ICML representative among the three joint invited speakers and spoke on "Reinforcement Learning: How Far Can It Go?"; Ron Kohavi, who was the second ICML invited speaker, discussing "Crossing the Chasm: From Academic Machine Learning to Commercial Data Mining"; and David Spiegelhalter, who was the jointly invited banquet speaker, describing "2.5 Millennia of Directed Graphs."
- the advisory committee for their suggestions regarding the program committee and the invited speakers.
- the program committee for their efforts initially reviewing about a dozen submissions each and then participating in in-depth discussions regarding which should be accepted.
- the organizers of the COLT, UAI, and ILP conferences for all their efforts spent coordinating our collocated meetings, and all the invited speakers and technical-paper presenters that these communities provided.
- Carol Hamilton of the American Association of Artificial Intelligence (AAAI) for her invaluable help coordinating AAAI '98 with ICML '98, and to AAAI in general for publicizing ICML '98, for processing ICML's advance registrations, for the organization of several ML-related workshops and tutorials, and for the use of the Madison convention center for some of the ICML '98 events.
- those sponsors (listed below) who provided financial support, allowing for reduced registrations fees and providing partial travel support to some needy graduate students and the invited speakers.
- the organizers of the joint AAAI/ICML workshops (listed below) and to the presenters of ML-related AAAI tutorials.

- Jacques Girard and Patricia Danek of the University of Wisconsin Business School, where the bulk of the conference's events were held, and to Maureen Sundell of the Wisconsin Office of Conference Services for their excellent help with local arrangements.
- Bradley Schwarzhoff for ably serving as a conference assistant, Laura Cuccia for secretarial help, and Sheila Beattie, Virginia Werner, Marie Johnson, Margaret Roth, and Benjamin Griffiths for processing much paperwork related to financial and legal matters.
- all the student volunteers who served before and during the conference, especially Tina Eliassi-Rad, Daniel Shiovitz, Chongmeng Chow, and Carolyn Alex.
- Morgan Kaufmann Publishers for distributing the volume; to Professional Book Center for producing it, and offering ICML contributors an experimental option to deliver their finished papers as PostScript files via the Internet (more than half of them did so); and to Steve Reiter of the United States Geological Service for his extraordinary help in locating and obtaining the cover image.

Jude Shavlik

Acknowledgments

The financial support of the following organizations is greatly acknowledged:

- Daimler-Benz Research and Technology
- United States Office of Naval Research
- Microsoft Research
- AT&T Research
- NEC Research Institute
- University of Wisconsin
- American Association of Artificial Intelligence

Advisory Committee

Ivan Bratko	Leslie Pack Kaelbling	Lorenza Saitta
William Cohen	Ron Kohavi	Derek Sleeman
Thomas Dietterich	Tom Mitchell	Paul Utgoff
Douglas Fisher	Stuart Russell	

Program Committee

David Aha	Diana Gordon	Filippo Neri
Christopher Atkeson	David Heckerman	David Opitz
Shumeet Baluja	Lisa Hellerstein	Foster Provost
Andrew Barto	Lawrence Hunter	Celine Rouveirol
Richard Belew	Jeffrey Jackson	Lorenza Saitta
Ivan Bratko	George John	Claude Sammut
Eric Brill	Michael Jordan	Dale Schuurmans
Carla Brodley	Leslie Pack Kaelbling	Satinder Singh
Claire Cardie	Dennis Kibler	Derek Sleeman
Richard Caruana	Ron Kohavi	Salvatore Stolfo
Corinna Cortes	Igor Kononenko	Richard Sutton
Gerald DeJong	David Lewis	Prasad Tadepalli
Luc De Raedt	Michael Littman	Sebastian Thrun
Thomas Dietterich	Richard Maclin	Paul Utgoff
Pedro Domingos	Sridhar Mahadevan	Manuela Veloso
Usama Fayyad	Maja Mataric	Grace Wahba
Douglas Fisher	Tom Mitchell	Stefan Wrobel
Nir Friedman	Andrew Moore	
Zoubin Ghahramani	Stephen Muggleton	

Workshops (Joint with AAAI '98)

Developing ML Applications: Problem Definition, Task Decomposition, and Technique Selection
(Robert Engels, Floor Verdenius, and David Aha)

Learning for Text Categorization
(Mehran Sahami, Mark Craven, Thorsten Joachims, and Andrew McCallum)

Predicting the Future: AI Approaches to Time-Series Analysis
(Andrea Danyluk, Thomas Fawcett, and Foster Provost)

CONTRIBUTED PAPERS

Query Learning Strategies using Boosting and Bagging

Naoki Abe Hiroshi Mamitsuka

Theory NEC Laboratory, RWCP*

c/o NEC C& C Media Research Laboratories

4-1-1 Miyazaki, Miyamae-ku, Kawasaki 216-8555 JAPAN

{abe, mami}@ccm.cl.nec.co.jp

Abstract

We propose new query learning strategies by combining the idea of query by committee and that of boosting [Sch90, FS95] and bagging [Bre94]. Query by committee is a query learning strategy which makes use of a randomized component learning algorithm and works by querying the function value of a point at which the predictions made by many copies of the component algorithm are maximally spread. The requirement of query by committee on the component algorithm that it be an ideal randomized algorithm makes it hard to apply in practice when we have only a moderately performing deterministic algorithm. To address this issue, we borrow the ideas of boosting and bagging, which are both techniques to enhance the performance of an existing learning algorithm by running it many times on a set of re-sampled data and combining the output hypotheses to make a prediction by (weighted) majority voting. We propose two query learning methods, query by bagging and query by boosting, which select the next query point by picking a point on which the (weighted) majority voting by the obtained hypotheses has the least margin. We empirically evaluate the performance of these methods on a wide range of real world data. Our experiments show that, when using C4.5 as the component learning algorithm and run on data sets in UCI Machine Learning repository, both query learning methods significantly improve data efficiency as compared to both C4.5 itself and boosting applied on C4.5. A typical increase in data efficiency achieved was 2 to 4-fold.

1 Introduction

Query learning is a sub-area of machine learning attracting increasing attention both in theory and in practice with the expectation that it may bring down both computational and sample complexities that plague passive learners. (c.f. [LC94, CS94, LG94]) For example, there is a rich body of work on the algorithmic approach to query learning as initiated by Angluin's query learning model [Ang87]. Another promising approach is the Bayesian or information theoretic approach to query learning [PK95, SOS92], in which a query learner tries to maximize the information gain on each query. Of the latter approach, 'query by committee' [SOS92] is an especially attractive and general query learning strategy with theoretical performance guarantee. In the present paper, we propose new variants of query by committee, which we call 'query by boosting' and 'query by bagging,' by combining query by committee with the techniques of boosting and bagging.

'Query by committee' [SOS92] is a query learning strategy which makes use of many copies of an ideal randomized learning algorithm. More concretely, it uses a number of copies of Gibbs algorithm (a randomized algorithm that picks a hypothesis from a given hypothesis class according to the posterior distribution and predicts according to it) and queries the function value of a point at which their predictions are maximally spread. The idea is that, by choosing a query point with maximum uncertainty of estimation of its function value, the information gain can be maximized. Indeed, there is a theoretical guarantee of the near-optimality of the data efficiency of this method, but it is based on the assumption that the component learning algorithm is Gibbs algorithm. This assumption poses two problems when one tries to apply this technique in practice: One is the problem of computational complexity, because Gibbs algorithms for interesting hypothesis classes tend to be computationally intractable. The other is that it cannot be applied on

*Real World Computing Partnership

a deterministic component learning algorithm. The two methods we propose in the present paper, ‘query by boosting’ and ‘query by bagging,’ are motivated to address these two issues.

‘Boosting’ and ‘bagging’ are both techniques to enhance the performance of an existing learning algorithm by running it many times on a set of re-sampled data and combining the output hypotheses to make a prediction. Bagging, due to Brieman [Bre94], is the simpler of the two, and it works by re-sampling from the input data with the same (uniform) distribution and its final hypothesis is obtained by taking majority vote over the predictions of the output hypotheses. Boosting¹ [Sch90, FS95] is a more complicated method that can be used to boost the performance of a relatively weak learning algorithm by use of sophisticated re-sampling on the training data. It does so by repeatedly re-sampling on the input training data, with the sampling distribution varied each time so as to focus more and more on the part of the training data on which the previously obtained hypotheses did poorly on. The final prediction of boosting is made by taking a weighted majority (or average) of the predictions of all the hypotheses thus obtained.

As noted earlier, one of the weakness of query by committee is that it cannot be applied on a deterministic component algorithm. If the component learning algorithm we have available is deterministic, the idea of bagging offers a natural alternative; namely apply bagging to obtain a set of hypotheses, let these hypotheses predict on a set of candidate points, and pick the point on which the predictions have the largest variance. When making a prediction, predict by majority vote over all the hypotheses. Since query by bagging introduces randomness in the form of re-sampling from the input data, it can be used on a component algorithm that is deterministic.

When the learning problem of interest is sufficiently complex, efficient implementation of Gibbs algorithm is not possible. If such is the case and the best known learning algorithm does not have a very good performance, then it makes sense to use boosting to enhance its performance. Recall that the most notable characteristic of boosting is its tolerance on the performance of the component learning algorithm. Thus, appropriately combining the idea of boosting and query by committee, we may obtain a query learning method that is tolerant on the performance of the component learning algorithm.

Recent experimentation using boosting has shown a remarkable fact (e.g. [DSS92]) that even after boosting

has achieved perfect prediction on the training data, it keeps boosting its predictive performance on unseen data. This seemingly contradicts known facts about over-learning, but recently Schapire *et al* [SFBL97] have given an account of this fact. That is, even after realizing perfect predictive performance on the training data, boosting keeps increasing its confidence of prediction, or more specifically the difference between the total weight assigned to the correct prediction and that assigned to a wrong prediction. (This is called the ‘margin’ of the prediction.) In their paper, they prove that a hypothesis having a larger margin on the training data performs better on unseen data as well. Based on this observation, the method we propose here, query by boosting, selects as the next query a point on which the margin obtained by the boosting algorithm is minimum, and attempts to maximize the uncertainty of prediction and hence the information gain on each query.

We conducted experiments using real world data to evaluate the performance of the proposed query learning methods. In particular, we tested them on a large part of the UCI Machine Learning data repository, using Quinlan’s C4.5 as the component algorithm. Here we note that testing query learning algorithms on these databases is not possible in a strict sense, since not all the query points can be answered. We therefore used our query strategies as methods of *selective sampling* to pick more informative queries from a fixed set of training data. (c.f. [LG94]) On almost all the data sets we tested these learning methods, both query by boosting and query by bagging achieved significant increase in data efficiency as compared to both C4.5 and boosting applied on C4.5. The increase in data efficiency measured by the data size required by the query learning methods to reach the same accuracy achieved by C4.5 (near the end of the data set) was anywhere from 2 to 5-fold. As compared to boosting applied on C4.5, the increase in data efficiency of the query methods was 2 to 4-fold on most data sets.

On one of the eight data sets above, tic-tac-toe, we ran analogous experiments using a different component learning algorithm – a randomized version of a weighted majority prediction algorithm for learning n -ary relations proposed in [ALN95] called WMP1. In addition to the two query methods, we also tested the original query by committee method, as the component algorithm is now randomized. It was found that, with randomized WMP1 as the component algorithm, both query by boosting and query by bagging performed better than query by committee.

¹Boosting was first discovered by Schapire [Sch90] in the context of proving the equivalence of ‘weak learnability’ with the strong PAC learnability. It was subsequently improved by Freund [Fre90], and Freund and Schapire [FS95].

Algorithm: Query-by-Committee(QBC)

Notation: In general, we use S'_i to denote the unlabeled sample corresponding to S_i .

Input: Number of trials: N

Randomized component learning algorithm: A

Number of times A is called: T

Number of query candidates: R

A set of query points: Q

Initialization: $S_1 = \langle x_1, f(x_1) \rangle$ for random x_1

For $i = 1, \dots, N$

1. Run A on S_i T times to obtain h_1, \dots, h_T .

2. Randomly generate a set of R points $C \subset Q \setminus S'_i$ with respect to uniform distribution over $Q \setminus S'_i$.

3. Pick a point $x^* \in C$ split most evenly: $x^* = \arg \min_{x \in C} ||\{t \leq T | h_t(x) = 1\} - \{t \leq T | h_t(x) = 0\}||$

4. Query the function value at x^* and obtain $f(x^*)$.

5. Update the past data as follows

$S_{i+1} = \text{append}(S_i, \langle x^*, f(x^*) \rangle)$

End For

Output: Output as the final hypothesis:

$h_{fin}(x) = \arg \max_{y \in Y} |\{t \leq T | h_t(x) = y\}|$

where h_t are hypotheses of the final (N -th) stage

Figure 1: Query by Committee (QBC)

Algorithm: Query-by-Bagging(QBag)

Input: Number of trials: N

Component learning algorithm: A

Number of times re-sampling is done: T

Number of query candidates: R

A set of query points: Q

Initialization: $S_1 = \langle x_1, f(x_1) \rangle$ for random x_1

For $i = 1, \dots, N$

1. By resampling according to uniform distribution on S_i , obtain sub-samples S'_1, \dots, S'_T each of size m .

2. Run A on each sub-sample and obtain h_1, \dots, h_T .

3. Randomly generate a set of R points $C \subset Q \setminus S'_i$ with respect to uniform distribution over $Q \setminus S'_i$.

4. Pick a point $x^* \in C$ split most evenly: $x^* = \arg \min_{x \in C} ||\{t \leq T | h_t(x) = 1\} - \{t \leq T | h_t(x) = 0\}||$

5. Query the function value at x^* and obtain $f(x^*)$.

6. Update the past data as follows

$S_{i+1} = \text{append}(S_i, \langle x^*, f(x^*) \rangle)$

End For

Output: Output as the final hypothesis:

$h_{fin}(x) = \arg \max_{y \in Y} |\{t \leq T | h_t(x) = y\}|$

where h_t are hypotheses of the final (N -th) stage

Figure 2: Query by Bagging (QBag)

2 Query Learning Methods

2.1 Query by Committee

We briefly describe the original query by committee method, generalized to use an arbitrary randomized component algorithm. At any point in time, query by committee runs the component algorithm on the past data a number of times to obtain many hypotheses. It picks the next query point by choosing from among a set of randomly generated candidate points a point such that the predictions by the hypotheses are split most evenly. The details are given in Figure 1. Here, if Q is a pre-determined set of points on which the function values can be obtained, then the algorithm as described is a method of selective sampling. If, on the other hand, Q is set to the entire domain, then it is a genuine query learning algorithm, which is free to choose any point in the domain as a query point.

2.2 Query by Bagging

'Bagging'[Bre94] re-samples from the input sample with a fixed distribution, and the final hypothesis is obtained by averaging the outputs of the hypotheses thus obtained. This method is based on the idea that prediction error consists of the 'bias,' which is the estimation error necessitated by the input data size, and the 'variance' which is due to the statistical variation existing in the specific data. The claim is that bagging can isolate the two factors and can minimize the

variance component of the error. Query by bagging is like query by committee, except it applies bagging on the input sample and picks as the next query point a point at which the predictions of the hypotheses are most evenly split. The details of query by bagging are also given in Figure 2.

2.3 Query by Boosting

We will now describe the query by boosting method in detail. In query by boosting, we pick as the next query point a point at which the weighted voting of the final hypothesis obtained by boosting the component learning algorithm has the least 'margin.' When the target function is 0,1-valued, this means that the query point is one for which the difference between the total weight for the value 1 and that for 0 is minimum among all candidate points. We give the details of this procedure in Figure 3, where we also supply the details of AdaBoost [FS95] for completeness.

Note that the original query by committee, query by bagging, and query by boosting form a natural progression. In query by committee, all the samples are *identical*, and the variance of the component algorithm's predictions is taken with respect to the randomness that exists within the component algorithm. In query by bagging, subsamples are obtained from the input sample using an *identical* distribution, and the variance of the component algorithm's predictions is with respect to the randomness in re-sampling. In

Algorithm: Query-by-Boosting(QBoost)**Input:** Number of trials: N Component learning algorithm: A Number of times re-sampling is done: T Number of query candidates: R A set of query points: Q **Initialization:** $S_1 = \langle x_1, f(x_1) \rangle$ for random x_1 **For** $i = 1, \dots, N$ 1. Run AdaBoost on input (S_i, A, T) and get:

$$h_{fin}(x) = \arg \max_{y \in Y} \sum_{h_t(x)=y} \log \frac{1}{\beta_t}$$

2. Randomly generate a set of R points $C \subset Q \setminus S'_i$ with respect to uniform distribution over $Q \setminus S'_i$.3. Pick a point $x^* \in C$ with the minimum margin:

$$x^* = \arg \min_{x \in C} \left| \sum_{h_t(x)=0} \log \frac{1}{\beta_t} - \sum_{h_t(x)=1} \log \frac{1}{\beta_t} \right|$$

4. Query the function value at x^* and obtain $f(x^*)$.

5. Update the past data as follows

$$S_{i+1} = \text{append}(S_i, \langle x^*, f(x^*) \rangle)$$

End For**Output:** Output h_{fin} in the last stage as the output.**Subroutine: AdaBoost [FS95]****Input:** Sample: $S = \langle (x_1, y_1), \dots, (x_i, y_i), \dots, (x_m, y_m) \rangle$ (Here, assume $\forall y_i \in Y = \{0, 1\}$.)Component learning algorithm: A Number of times re-sampling is done: T **Initialization:** $\forall i \leq m, D_1(x_i) = \frac{1}{m}$ **For** $t = 1, \dots, T$ 1. Run A on a sample of size m generated w.r.t. D_t .2. Let its output hypothesis be h_t .3. Compute its error rate ϵ_t by:

$$\epsilon_t = \sum_{h_t(x_i) \neq y_i} D_t(x_i)$$

4. Calculate β_t by $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ 5. Update the re-sampling distribution D_{t+1} :

$$D_{t+1}(x_i) = \frac{D_t(x_i) \cdot \beta_t}{Z} \quad \text{if } h_t(x_i) = y_i$$

$$D_{t+1}(x_i) = D_t(x_i) \quad \text{otherwise}$$

(Here Z is a normalization constant satisfying

$$\sum_{i=1, \dots, m} D_{t+1}(x_i) = 1.)$$

Output: Output as the final hypothesis:

$$h_{fin}(x) = \arg \max_{y \in Y} \sum_{h_t(x)=y} \log \frac{1}{\beta_t}$$

Figure 3: Query by boosting (QBoost)

name	# ex.	# attributes		missing values
		disc.	cont.	
liver-disorders	345	-	6	-
ionosphere	351	-	34	-
house-votes-84	435	16	-	o
wdbc	569	-	32	-
crx	690	9	6	o
breast-cancer-wisconsin	699	9	-	o
pima-indians-diabetes	768	-	8	-
tic-tac-toe	958	9	-	-

Table 1: The eight data sets used in our experiments.

query by boosting, the re-sampling distribution itself is changed depending on the properties of the obtained hypotheses, and the variance of the component algorithm's predictions is measured with respect to the uncertainty involved in weighted voting by the various hypotheses.

3 Experimental procedures

We evaluate the proposed query learning methods on the learning problem for concepts (or 0,1-valued functions) over a number of attributes, which are either binary, discrete or numerical. A special case of this is when all the attributes are discrete, and the target function can be regarded as an n -ary relation over n finite sets. In our experiments, we use existing data sets for training and test data, without an explicitly defined target function. Since it is not possible to use query learning algorithms genuinely as query learners in this setting, we use them as methods for *selective sampling*, that is, ways to select a smaller set of more effective data from a large data set.

The data sets we used in our experiments were borrowed from the machine learning data repository of University of California at Irvine.² Of the large number of data sets available from the repository, we selected 8 (not all) data sets satisfying the following conditions: (1) The target function is 0,1-valued; (2) The data size is moderate (more than 300 and less than 1,000); Table 1 summarizes the data sets we selected and their basic characteristics.

On these data sets, we compared the performance of C4.5, boosting applied on C4.5, query by boosting applied on C4.5, and query by bagging applied on C4.5. For each data set, we performed 10-fold cross validation, with one-tenth of the available data (selected randomly) reserved as the test data and the rest used as the training data, or query data. For each of the 10

²This data set, abbreviated as the ICI ML repository in what follows, is available at URL address: "http://www.ics.uci.edu/~mllearn/MLRepository.html"

pairs of training and test data sets, we averaged the results over two randomized runs, a total of 20 runs for each data set.³

The query learning algorithms are used to pick the next query point from the training (query) data without replacement and are tested using the (separate) test data. When the specified number of candidates exceeded what is left of the training data, we went on with as many candidates as there were left. On one occasion, we also examined their predictive performance on the *query* data, from which query learners have selected a subset to learn from, instead of using the separate test data.

Finally, the parameters T and R in all the query learning methods were set at $T = 20$ and $R = 100$ in all of our experiments.

4 Experimental Results

We now discuss the results of our experiments on the UCI Machine Learning Repository. Figure 5 plots the learning curves obtained for the four learning methods on each of the eight data sets. Each graph plots the predictive accuracy (in percentage) of the four learning methods measured using the separate test data at every 50 trials. It is clearly seen from these graphs that in *all* eight data sets, the two proposed query learning methods achieve significant improvement in data efficiency as compared to C4.5. One can see that at very early stage in learning, say around 50 to 150 trials depending on the data set, the prediction accuracy of the query learning methods reaches a level that is achieved by C4.5 only towards the end of the data set. Table 2 gives concrete figures that quantify this observation. Here, 'the target error rate' was calculated using the error rate of C4.5 in the last 100 trials.⁴ Then, we checked to see how many trials it took for all four methods to reach that error rate. In parentheses, we also exhibit the ratio of the number of trials required by each of the methods to that of C4.5. One can see that typically the data efficiency is improved by a factor of 2 to 4.

The speed-up achieved by the two query learning methods compared against boosting applied on C4.5 is less dramatic but still significant. From the graphs, one can see that on five of the eight data sets, namely breast-cancer-wisconsin, tic-tac-toe, ionosphere, house-votes-84 and wdbc, the advantage of the query methods over boosting is clear, while on the

other three it is less obvious. These three data sets, crx, liver-disorders, and pima-indians-diabetes appear to have a common feature: That a certain level of accuracy is achieved with relatively few examples, but from then on the accuracy is hardly improved as the data size increases. It may be that the target function of these data sets is sufficiently noisy that no learning method can break this barrier. The increase in data efficiency achieved by the query learning methods in comparison to boosting is summarized in Table 3, similarly as before.

All the evaluation discussed thus far has been based on the prediction accuracy measured using *test* data, which are disjoint from the training data or the query data from which the query learning methods selected query points. As we remarked earlier, this is selective sampling and not genuine query learning. If we measure the prediction accuracy of query learning algorithms with respect to the *query* data, then this would translate to a genuine query learning scenario, except the function being learned is solely defined by the query data, only on those points that are in the data. We took this view point and examined the learning curves for the four methods with respect to this measure. Figure 6 plots these learning curves for the eight data sets as before. One can more clearly see the effect of query learning here – with respect to all but one data set (pima-indians-diabetes), the accuracy of the two query learning methods rise much faster than either C4.5 or boosting on C4.5., typically achieving an increase in data efficiency of factor 3 to 6.

On one of the eight data sets, tic-tac-toe, we ran the analogous experiments as above using a randomized version of WMP1 as the component learning algorithm. Figure 4 plots the prediction accuracy achieved by each of the five methods at the end of every 50 trials. Note that query by committee can now be applied because we use a randomized component algorithm. Here much of the tendency observed using C4.5 carries over. Notice, however, that here the two proposed methods, query by boosting and query by bagging, out-perform query by committee. Also, in this case query by boosting seems to do better than query by bagging, at least for a wide range of data sizes. The relative performance of the competing query learning methods appear to depend on the component learning algorithm (and the learning problem). Note further that boosting and the query methods applied on WMP1 achieve much higher accuracy than those applied on C4.5 on this particular problem. Interestingly, WMP1 itself does not have a higher accuracy than C4.5, but both boosting and query by boosting applied on WMP1 are significantly more effective than those applied on C4.5.. This observation suggests that on component algorithms and problems on which boosting is effective, query by boosting may do better than

³The results involving WMP1 were obtained by averaging over 10 runs, not 20 runs.

⁴For this calculation, we fed a randomly chosen test example after each trial, and the prediction error of the current trial was calculated by the average prediction error over the last 50 test trials.

name	query by bagging	query by boosting	boosting	C4.5	total size	target error rate (C4.5)
liver-disorders	86(0.30)	96(0.34)	108(0.38)	286(1.0)	310	0.3685
ionosphere	91(0.39)	97(0.41)	143(0.61)	236(1.0)	315	0.0935
house-votes-84	65(0.21)	72(0.24)	145(0.48)	303(1.0)	391	0.0465
wdbc	82(0.26)	88(0.28)	208(0.66)	314(1.0)	512	0.054
crx	64(0.50)	100(0.79)	119(0.94)	127(1.0)	621	0.171
breast-cancer-wisconsin	86(0.40)	83(0.39)	209(0.98)	213(1.0)	629	0.072
pima-indians-diabetes	67(0.44)	63(0.41)	81(0.53)	152(1.0)	691	0.2895
tic-tac-toe	236(0.39)	243(0.40)	308(0.51)	609(1.0)	862	0.1445

Table 2: Data efficiency increase achieved with respect to C4.5

name	query by bagging	query by boosting	boosting	C4.5	total size	target error rate (boosting)
liver-disorders	111(0.86)	126(0.98)	129(1.0)	-	310	0.3305
ionosphere	121(0.50)	119(0.49)	243(1.0)	-	315	0.073
house-votes-84	71(0.34)	136(0.65)	210(1.0)	366(1.74)	391	0.04
wdbc	97(0.32)	130(0.43)	300(1.0)	506(1.69)	512	0.0455
crx	86(0.60)	140(0.97)	144(1.0)	-	621	0.146
breast-cancer-wisconsin	103(0.34)	92(0.31)	301(1.0)	391(1.30)	629	0.0495
pima-indians-diabetes	99(0.56)	191(1.09)	176(1.0)	-	691	0.2475
tic-tac-toe	438(0.52)	517(0.62)	836(1.0)	-	862	0.053

Table 3: Data efficiency increase achieved with respect to boosting

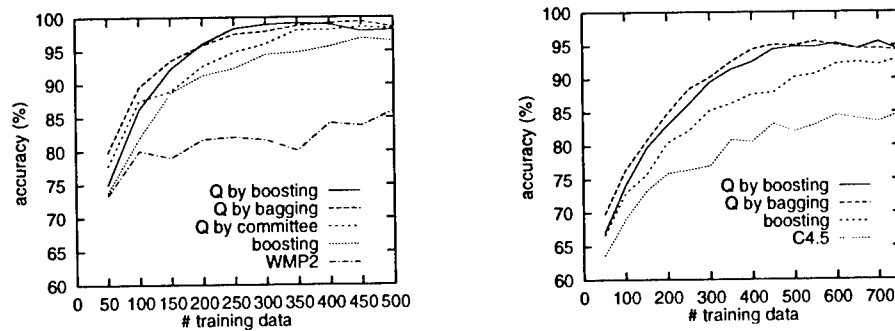


Figure 4: Prediction accuracy on test data on tic-tac-toe. Left: Using WMP1 as the component algorithm, Right: Using C4.5 as the component algorithm.

the other query learning methods as well.

The time complexity of all three query learning methods we considered is of the order $O(NTR \cdot F(N))$, where $F(N)$ is the time complexity of the component algorithm when run on an input sample of size N . This is a tractable but significant increase in computation cost as compared to the component algorithm. The judgement of whether the data efficiency brought about by these methods justifies the additional computational burden would depend on the exact application under consideration. Also note that both query by committee and query by bagging are parallelizable with respect to T and R , but query by boosting is parallelizable only with respect to R , and not T . Thus, only when query by boosting buys significantly more data efficiency, would it be the method of choice.

5 Concluding Remarks

We proposed two variants of query by committee that can be applied on an arbitrary component algorithm, be it deterministic or randomized, by incorporating the ideas of boosting and bagging. Experiments on data sets from the UCI Machine Learning repository demonstrated that, when using them with C4.5 as the component algorithm, the proposed query learning methods achieve significant increase in data efficiency as compared to both C4.5 and boosting applied on C4.5. On one of the data sets which can be cast as an n -ary learning problem, we tested these methods using a randomized weighted majority prediction algorithm for n -ary relations as the component algorithm, and found that the proposed methods performed better than query by committee. In the near future, we plan to carry out more systematic evaluation to verify the robustness of the proposed query methods on the choice of the component algorithm and the learning problem.

Acknowledgement

We thank Dr. S. Goto, Dr. S. Doi of NEC and Dr. K. Takada of NIS for their support. We would also like to thank those involved with the UCI Machine Learning repository for making the data sets available.

References

- [ALN95] N. Abe, H. Li, and A. Nakamura. On-line learning of binary lexical relations using two-dimensional weighted majority algorithms. In *Proc. 12th Int'l. Conference on Machine Learning*, July 1995.
- [Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, November 1987.
- [Bre94] L. Breiman. Bagging predictors. Technical Report 421, University of California at Berkeley, 1994.
- [CS94] Mark W. Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning: Proceedings of the 11th International Conference*, pages 37–45, 1994.
- [DSS92] H. Drucker, R. Schapire, and P. Simard. Improving performance in neural networks using a boosting algorithm. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1992.
- [Fre90] Y. Freund. Boosting a weak learning algorithm by majority. In *Proc. 3rd Annu. Workshop on Comput. Learning Theory*, pages 202–216. Morgan Kaufmann, San Mateo, CA, 1990.
- [FS95] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory (EuroCOLT'95)*, pages 23–37, 1995.
- [LC94] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning: Proceedings of the 11th International Conference*, pages 148–156, 1994.
- [LG94] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. *Proceedings of 17th Annual International ACM-SIGIR Conference of Research and Development in Information Retrieval*, pages 3–12, 1994.
- [PK95] G. Paass and J. Kindermann. Bayesian query construction for neural network models. In *Advances in neural information processing systems 7*, pages 443–450, 1995.
- [Sch90] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
- [SFBL97] R. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, pages 322–330, 1997.
- [SOS92] H. S. Seung, M. Oppor, and H. Sompolinsky. Query by committee. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 287–294. ACM Press, New York, NY, 1992.

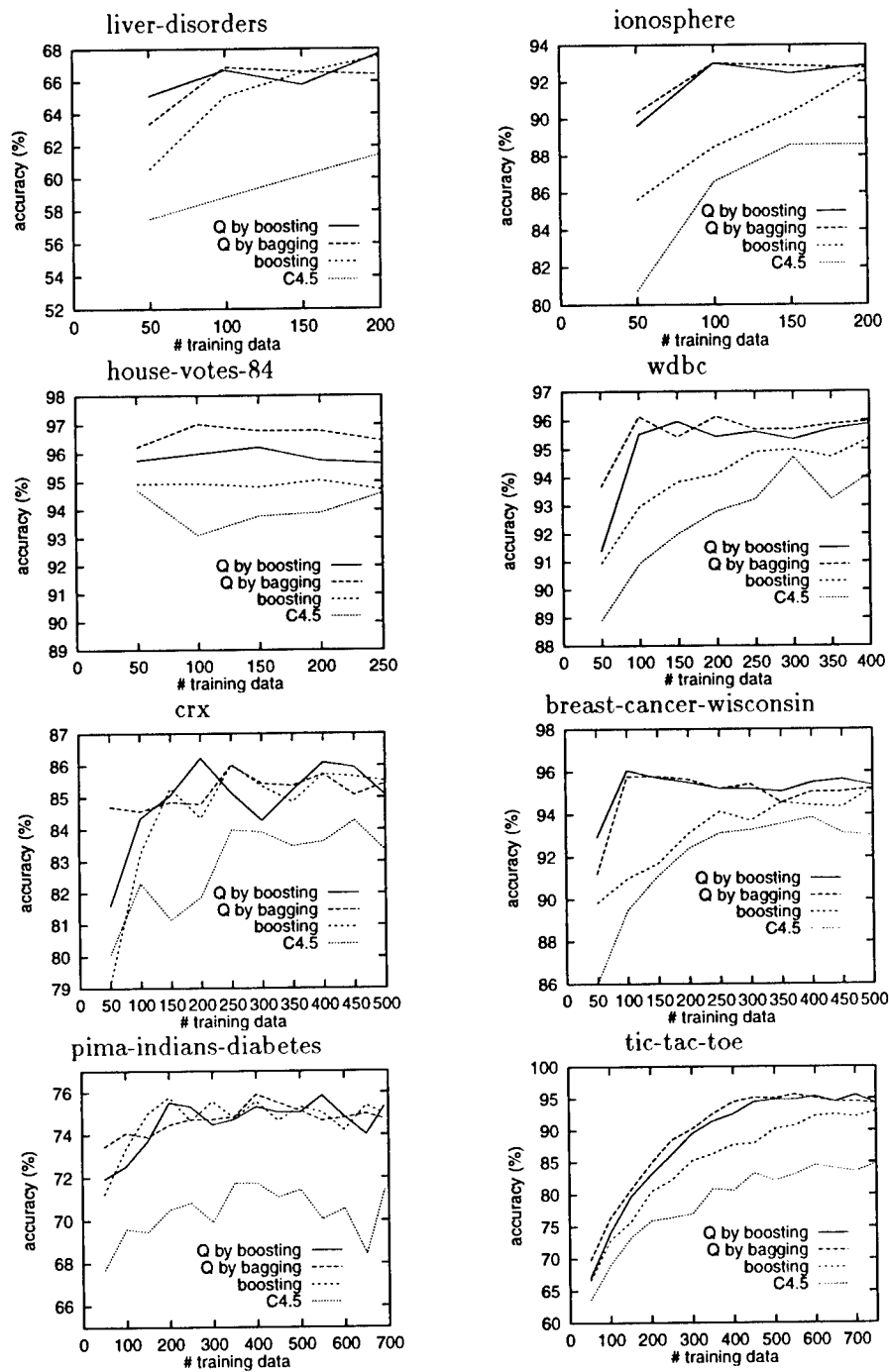


Figure 5: Learning curves for four learning methods on the UCI ML Repository.

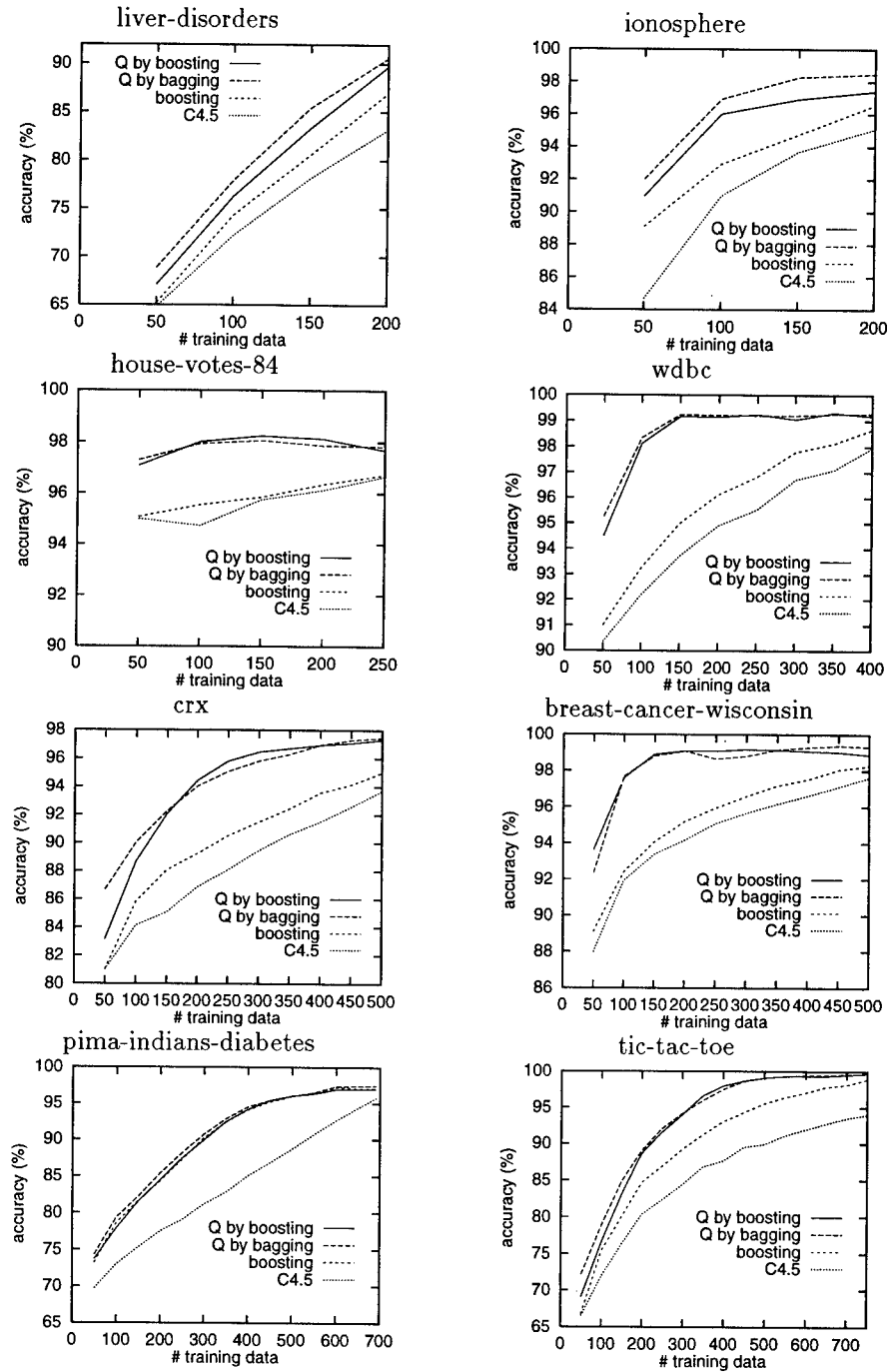


Figure 6: Learning curves on 'query data' for four learning methods on the UCI ML repository.

Genetic Programming and Deductive-Inductive Learning: a Multi-strategy Approach

Ricardo Aler, Daniel Borrajo, Pedro Isasi

Departamento de Informática, Universidad Carlos III de Madrid

28911 Leganés (Madrid), Spain

email: {aler@inf,dborrajo@ia,isasi@ia}.uc3m.es

Abstract

Genetic Programming (GP) is a machine learning technique that was not conceived to use domain knowledge for generating new candidate solutions. It has been shown that GP can benefit from domain knowledge obtained by other machine learning methods with more powerful heuristics. However, it is not obvious that a combination of GP and a knowledge intensive machine learning method can work better than the knowledge intensive method alone. In this paper we present a multi-strategy approach where an analytical and inductive approach (HAMLET) and an evolutionary technique based on GP (EvoCK) are combined for the task of learning control rules for problem solving in planning. Results show that both methods complement each other, supplying to the other method what the other method lacks and obtaining better results than using each method alone.

1 INTRODUCTION

Genetic Programming (GP) is a machine learning technique based on a search over a huge state space [Koza and Rice, 1991]. Therefore, as any search method, it can be defined in terms of three elements: an initial state, a set of operators, and a heuristic function (called fitness function). GP expands the ideas of Genetic Algorithms by using structured representations (trees). The use of this type of representation is more appropriate for solving symbolic tasks than Genetic Algorithms.

One of such tasks consists on learning control knowledge for problem solving. Problem solving can also be described in terms of a search in another state space than the one of GP. Traditional approaches use domain independent planners for generating plans [Blum and Furst, 1995, Penberthy and Weld, 1992]. PRODIGY, an architecture for planning and learning that uses a means-ends analysis nonlinear planner, is one of such systems [Veloso *et al.*, 1995]. However, planning becomes impractical for large problems. In order to gain efficiency, PRODIGY must be supplied with domain-dependent search control knowledge which can be applied at decision points in the planning reasoning cycle. This control knowledge has the form of control rules, as further explained later on.

In this type of tasks, the use of all available domain knowledge is essential for an efficient learning process. Classically, GP systems have only used domain knowledge for the fitness function. We propose the use of background knowledge coming from the use of a previous learning technique also in another two search elements [Aler *et al.*, 1998a]: first, the initial state will not be created randomly, but using control knowledge learned by another method, HAMLET in this case [Borrajo and Veloso, 1997]. Second, genetic operators will use knowledge in the form of examples, obtained as a sub-product of HAMLET learning process.

In [Aler *et al.*, 1998a] we have shown that GP obtains much better results in planning by using such background knowledge. The purpose of this paper is to show that a multi-strategy approach using GP and HAMLET works better than using each method alone. This multi-strategy approach can be seen as a combination of learning bias from different methods: GP and HAMLET. In this paper, we have used PRODIGY, but in the future other planners such as UCPOP or GRAPHPLAN might be used.

Section 2 explains the role of learning in planning. Section 3 describes our multi-strategy approach for learning in planning. Section 4 describes our experimental setup and the results obtained. Section 5 discusses these results, and presents the conclusions. Finally, Section 6 surveys related work.

2 THE LEARNING TASK

The learning task can be stated as: given a set of traces belonging to problems solved by PRODIGY in a particular planning domain, induce a set of control rules that perform well in that planning domain. Control rules help PRODIGY to make decisions at several points in its search process. If there are no applicable control rules in a decision point, PRODIGY will make a default decision. It has five kinds of decision points:¹

- Select, prefer or reject a goal from the set of pending goals.
- Select, prefer or reject an operator to achieve a goal.
- Select, prefer or reject a binding for the chosen operator.
- Choose whether to apply an instantiated applicable operator or to subgoal on an unachieved goal.
- Select, prefer or reject an instantiated operator from the set of applicable instantiated operators.

Figure 1 shows an example of a control rule for the blockworld domain. `current-goal`, and `true-in-state` are meta-predicates. The control rule says that if PRODIGY is working on trying to hold an object, `<object1>`, and this object is on top of another, `<object2>`, in the current state, then PRODIGY should select the operator `UNSTACK` and reject the rest of operators that could achieve the same goal.

```
(control-rule select-operators-unstack
  (if (and (current-goal (holding <object1>))
           (true-in-state (on <object1> <object2>))))
  (then select operator unstack))
```

Figure 1: Example of a control rule for making the decision of what operator to use.

¹HAMLET only generates selection control rules. In this article, GP will look just for that kind of control rules, so that it can be properly compared with HAMLET.

At every decision point, PRODIGY is in a particular search meta-state. Let *ME* be the set of all possible meta-states. Now, helping PRODIGY to take decisions can be stated as: for each possible decision (for example: `select goal (on x y)`) find a partition of *ME* into *ME+* (where the decision should be taken) and *ME-* (where the decision should not be taken). That is, control rules are actually classification rules: they partition the space of meta-states into those meta-states that belong to a possible decision and those that do not. And this looks like traditional machine learning concept induction, where classification rules have to be induced from a set of examples. In this case, it has the following characteristics:

- Several target concepts have to be learnt from the same data (set of traces). Not only there are different kinds of target concepts associated to each kind of decision (select operator, select goal, etc) but each kind of decision has several associated target concepts. For instance, there will be one target concept of the type select operator for each possible (operator, goal) pair of a particular domain.
- Target concepts will generally be disjunctive (that means that several control rules will be needed to represent a target concept).
- The representation of concepts is relational, so we are dealing with an ILP problem.

Therefore, when using GP, each individual will be a set of control rules, represented as a structure that will be explained in Section 3.2. A GP population is made of several such individuals.

3 A MULTI-STRATEGY APPROACH FOR LEARNING CONTROL KNOWLEDGE

In this section we will describe the architecture of the learning system, and define the learning behavior in terms of its three learning biases.

3.1 ARCHITECTURE OF THE LEARNING SYSTEM

The general architecture of our system consists of five blocks (as also shown in Figure 2). The main blocks are EVOCK ("Evolution of Control Knowledge") and HAMLET.

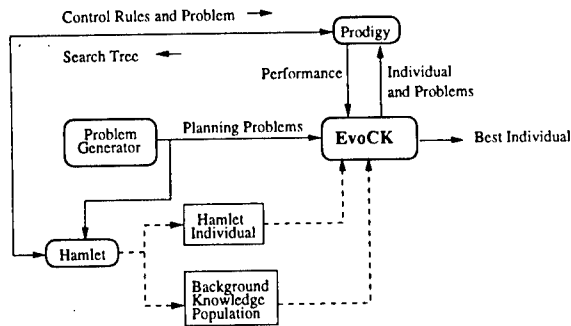


Figure 2: General Architecture of the multi-strategy approach.

EvoCK is the module that implements the GP paradigm adapted for evolving planning control rules. EvoCK is supplied with fitness cases generated by a problem generator. These fitness cases are planning problems generated at random by the problem generator. In order to evaluate individuals from the population with the fitness cases set, EvoCK tells PRODIGY to load the individual and try to solve each one of the fitness cases. Performance of this individual with these fitness cases is returned to EvoCK. HAMLET has a similar relation with PRODIGY and the problem generator but in this case the information returned by PRODIGY is the search tree that HAMLET will use to generalize and refine its control-rules.

EvoCK and HAMLET are weakly coupled in the following way. First, HAMLET is run to learn from a set of randomly generated problems. Then, two of its outputs are used as background knowledge for EvoCK: the set of rules learned by HAMLET (“HAMLET individual”) are used to seed the EvoCK initial population. Also, the HAMLET supplies a set of positive examples (“Background Knowledge Population”) that will be taken as input by one of the genetic operators (knowledge based crossover [Aler *et al.*, 1998a]). This will be explained in subsection 3.3.

When EvoCK gets to the maximum number of evaluations allowed for learning, it returns its best individual obtained so far. Although not shown in Figure 2, best individuals are tested with a different set of planning problems (also obtained from the problem generator) to check how well they have generalized from the training data.

In the next three sections, we describe the system by explaining its learning biases. These biases are classified following Utgoff [Utgoff, 1986] in language biases, exploration biases and evaluation biases.

3.2 THE LANGUAGE BIAS

Usually, in GP there are no constraints in the structure that is to evolve: any combination of functions and terminals will be valid and crossover points can be taken at any place in the individual. But, in our case, PRODIGY restricts what are valid structures and what are not. For instance, a meta-predicate like TRUE-IN-STATE² can only be passed as argument a goal like (on <x> <y>) but not an operator like PUT-DOWN. Other general constraints are imposed by the structure of the rule language itself (if <condition> then <action>, etc). In many cases this problem can be solved by achieving operational closure, that is, by allowing each function to accept any type of result [Koza and Rice, 1991]. However, this is not possible in this case, since PRODIGY fixes the structure of the language for representing control rules and feeding it with non-valid control rules would make it fail.

Therefore, we have chosen to constrain structures to PRODIGY-valid ones (in the literature, such structures are called “constrained structures” [Koza and Rice, 1991] or “strongly typed structures” [Montana, 1995]). In order to achieve it, the following three steps must be followed: create only valid structures, crossover points must be of the same type and mutation operators must take into account the type of the mutation point. The first step is achieved by using an special-purpose production grammar. An example of an individual generated by the grammar might be the one that appears in Figure 3. This individual consists of two control rules for the blocksworld domain. The first one checks whether there is a block with no other blocks on it and if the planner is trying to solve either putting that object on another object or having the robot arm hold a third different object. If both conditions succeed, then the planner will work next in the (on <object-1> <object-2>) goal. The other control rule says that if there is an object on the table and the system is trying to bind the pick-up operator, then it should be bound to that object.

3.3 THE EXPLORATION BIAS

The exploration bias includes everything related to the search policy: search operators, background knowledge to constrain the search, etc. The system uses

²Meta-predicates are functions that have access to PRODIGY meta-state. Therefore they can check whether a condition is true or not in the meta-state. For instance TRUE-IN-STATE tests if a particular condition is true in the current planning state


```
(list (rule (and (true-in-state (clear <object-1>))
  (some-candidate-goals
    (goals-list (on <object-1> <object-2>))
    (holding <object-3>)))
  (select-goal (on <object-1> <object-2>)))
(rule (true-in-state (on-table <object-1>))
  (select-bindings (pick-up-b <object-1> ))))
```

Figure 3: Example of EVOCK individual.

the traditional GP operators (crossover and mutation) and some others specially tailored for the learning task. The whole operator set is:

- **Copy:** reproduction without modification.
- **Xover:** traditional crossover. It takes two constrained structures and produces one constrained structure
- **Changing_mutation:** it chooses a mutation point, and changes the whole subtree by another randomly generated subtree. This mutation is equivalent to **Xover** with a randomly generated individual (as the second parent).
- **Xover_add:** some points in the evolving structure allow for lists of elements of the same kind (as, for instance, lists of goals). In those cases, crossover adds elements to the lists from the other parent, instead of replacing the whole list.
- **Chopping_off_mutation:** in those points where lists of elements of the same kind are allowed, it removes one of the elements.
- **Growing_mutation:** it adds a random subtree at those points where lists of elements of the same type are allowed. It is equivalent to **Xover_add** with a randomly generated individual (as the second parent).

All these operators are simple variations of genetic operators traditionally used in GP. The next two operators are specially tailored for this learning task.

- **Join:** it selects one variable in the control rule (like <object-1>) and substitutes it by any other variable in the control rule. The rationale behind this operator is that sometimes there are conditions in a rule that are not related with other conditions by common variables. Sometimes that is undesirable. For instance, if we have a control rule to pick-up an object <obj1> when some conditions are true, our experience says that many

of those conditions should refer to <obj1>. The **join** operator is a simple way of creating these references.

- **Up_the_hierarchy:** objects (the elements to which the planning operators are applied) in PRODIGY are organized in a tree-shaped type hierarchy. For instance, in logistics transportation planning domain, there are trucks and planes, which are both defined as carriers. This genetic operator would take a truck-typed variable in the left hand side of the rule and would substitute all its instances by a carrier-typed variable. Thus, the control rule would become more general.

The related specialization operators (i.e. **disjoin** and **down_the_hierarchy**) are not included in the operator pool; we are imposing a strong bias towards generalization. However, the system can still specialize by means of the other generic operators (mutation, etc).

Background knowledge can be introduced to the system in order to restrict the search. So far, we have used two kinds of background knowledge:

- Seeding the initial population with an individual coming from HAMLET.
- The early phase of HAMLET returns a set of positive and negative examples as a sub-product. Positive examples are those where PRODIGY made the right decision in the planning process. These positive examples can be easily transformed into control rules and then into GP individuals. Then, the crossover operator will be able to draw individuals from the background knowledge population instead of the evolving population (this is what we have called "knowledge-based crossover operator" [Aler *et al.*, 1998a]). In that way, background knowledge can be injected into the evolving population.

Finally, we use a steady state GP with a generational gap of 1. 2-tournaments are held for both selection and replacement. This has been shown experimentally to behave well.

3.4 THE EVALUATION BIAS

The evaluation bias concerns the preference criteria used by GP for selecting an individual over another, which is coded as a fitness function. In our case, we devised a hierarchical fitness function that contains the

following components [Aler *et al.*, 1998b, Aler *et al.*, 1998a]:

1. **Performance in fitness cases:** to maximize. How well the individual performs when PRODIGY tries to solve the training planning problems when guided by the individual (acting as a set of control rules). It will be explained later in more detail.
2. **Number of different variables:** to minimize. This fitness component is related to the same bias than the join operator. We want to have as many meta-predicates in the left hand side of the control rules inter-related by common variables as possible.
3. **Number of different true-in-state meta-predicates:** to minimize. The fewer true-in-state meta-predicates, the more general and faster will run the set of control rules.
4. **Number of different goals in some-candidate-goals meta-predicates:** to maximize. This meta-predicate returns true if at least one of its arguments is a candidate goal to be solved by the planner. So, the more goals has some-candidate-goals, in more cases it will be applicable and the more general it will be (although less compact).
5. **Number of different some-candidate-goals:** to maximize. Another way of making a rule more general is to get rid of unnecessary some-candidate-goals checking. This also makes it faster.
6. **Number of control rules:** To minimize. The fewer control rules, the faster will the individual solve the problems.
7. **Individual size (in nodes):** To minimize.

All individuals in the tournament set that have the same score in the first comparison will pass to the second one and so on. The rest will be dropped off the tournament. If more than one individual happens to pass the last comparison, the tournament winner is chosen randomly.

The first criteria, performance in fitness cases, was formerly computed by measuring how many steps of the solution of a given planning problem the individual managed to follow (solutions to all the planning problems were known by EVOCK in advance by letting PRODIGY solve those problems and storing the

search trees). However, although we obtained good results, we realized that an individual managing to follow many steps in the solution didn't guarantee that the individual would actually solve the problem. Therefore, we have decided to change it for a set of three new criteria:

- **Number of problems solved by PRODIGY being guided by the individual with a maximum node limit.** To maximize. This node limit is four times the amount of nodes that would be needed to solve the problem if PRODIGY could go straightforward to the solution.
- **Number of problems solved by the individual more efficiently than PRODIGY alone.** To maximize. Efficiency in this case means fewer nodes expanded.
- **Total number of nodes expanded by the individual.** To minimize.

In order to test an individual with these new criteria, it has to be loaded into PRODIGY. Then, PRODIGY will be run for each of the planning problems for learning (or fitness cases, in GP terminology). However, complex problems need to be given a high node limit if they are to be solved. As many such evaluations must be performed for each generation, only simple problems can be used for learning (otherwise the fitness function would take too long). This is another bias to take into account.³ However, [Borrajo and Veloso, 1997] shows empirically that training with simple problems is enough for learning control knowledge useful to solve more complex problems.

4 EXPERIMENTAL RESULTS

In order to test our multi-strategy approach, the following steps were carried out:

1. Hamlet was trained with 400 learning planning problems. Two domains were used: blocksworld and logistics. A set of control rules and a set of positive examples were obtained for each domain. They were used as background knowledge in the next step.
2. EVOCK was trained in the blocksworld and logistics with 192 and 188 learning planning problems

³ [Aler *et al.*, 1998b, Aler *et al.*, 1998a] was not constrained by this bias.

respectively. A population size of 2 was used. Certainly, a population size of 2 is not common in GP. Previous work [Aler *et al.*, 1998a] shows that using a bigger population seems to be good but results are not conclusive: the interaction between population size and seeding the initial population is not properly understood yet. In our case, the seeding individual (coming from HAMLET) is much better than the other initial individuals (randomly generated) therefore two things might happen: first, during the earlier generations the seeding individual would not be selected very often, so some time would be spent evaluating individuals that contain no knowledge. Second, if the seeded individual is much better than the randomly generated individuals, in the long term all members might contain similar genetic information to the seeded individual [Fraser and Rush, 1994]. In this paper a population of 2 has been chosen because in that way, we make sure that genetic operators will always act on individuals which contain knowledge and therefore, the impact of knowledge will be better controlled. In any case, we plan to carry out several experiments that will study the population size-population seeding interaction in detail. Performing crossover in such a small population is not meaningful, so standard crossover is not used in this paper. However, EvoCK can use it in general. Background knowledge from the previous step was used in the two ways described in subsection 3.3. As GP is a stochastic method, several experiments were carried out for each domain: 47 for the blocksworld and 54 for logistics. Each experiment ran for 100.000 evaluations. From each experiment, a set of control rules was obtained.

3. HAMLET was trained in a similar manner than EvoCK. HAMLET started with the sets of control rules obtained in step 1 and refined them with the rest of the learning problems used to train EvoCK. Two sets of control rules were obtained (one for each domain).
4. Finally the sets of control rules obtained by EvoCK and HAMLET were tested with a new set of problems (416 for the blocksworld and 347 for logistics) in the same conditions. Results are shown in Table 1. As EvoCK obtained one set of rules from each experiment, two quantities are shown: the number of problems solved by the best of all sets of control rules (along with the number of control rules for that individual) and the aver-

age number of problems solved over all sets.

Table 1: Results for PRODIGY, HAMLET and EvoCK in both the blocksworld and logistics domains.

	% Prob. Solved	Number of Rules	Average % P. Solv.
Blocksworld			
PRODIGY ALONE	21%		
HAMLET SEED	58%	12	
HAMLET	18%	13	
EvoCK (best indiv.)	87%	4	80%
Logistics			
PRODIGY ALONE	43%		
HAMLET SEED	52%	56	
HAMLET	46%	64	
EvoCK (best indiv.)	95%	19	65%

Table 1 shows that when HAMLET tries to refine and improve a set of control rules previously learned (HAMLET seed in Table 1), the percentage of test problems actually solved drops: in the blocksworld it goes from 58% to 18%, in logistics it gets from 52% to 46%. On the other hand, EvoCK improves the set of control rules given as seed for the initial population: 58% to 87% in the blocksworld and 52% to 95% in logistics. Next section comments on these results. It is also noticeable that EvoCK produces individuals with fewer control rules than the seeding individual (12 to 4 control rules in the blocksworld and 56 to 19 in logistics) hence returning more efficient individuals. In order to show that the control rules learned are general and useful for more complex problems, a breakdown of the results are displayed in Tables 2 and 3.

Table 2: Breakdown of the number of testing problems solved in the blocksworld by HAMLET and EvoCK according to the number of goals and of objects).

# Goals	# Objects	PRODIGY	HAMLET seed	HAMLET	EvoCK
50	50	0%	0%	0%	56%
20	50	6%	31%	4%	81%
20	20	6%	27%	4%	69%
10	50	21%	67%	19%	96%
10	20	15%	56%	15%	83%
10	15	31%	48%	15%	85%
5	50	15%	70%	2%	92%
5	20	15%	82%	18%	95%
5	15	40%	82%	35%	98%
5	10	50%	85%	60%	95%

Tables 2 and 3 show a breakdown of the number of problems solved by the different methods in the blocksworld according to problem complexity. This

Table 3: Breakdown of the number of testing problems solved in logistics by HAMLET and EVOCK according to the number of goals and of objects).

# Goals	# Objects	PRODIGY	HAMLET seed	HAMLET	EVOCK
50	50	0%	0%	0%	75%
20	50	3%	0%	0%	100%
20	20	7%	28%	0%	83%
10	50	13%	0%	0%	100%
10	20	20%	53%	33%	100%
10	15	20%	67%	47%	100%
10	10	7%	67%	40%	100%
5	50	42%	0%	0%	100%
5	20	58%	83%	67%	100%
5	15	42%	42%	67%	100%
5	10	25%	58%	67%	100%
5	5	33%	83%	92%	100%
2	50	90%	60%	20%	100%
2	20	90%	100%	100%	100%
2	15	90%	90%	100%	100%
2	10	90%	80%	90%	100%
2	5	100%	100%	100%	100%
2	2	100%	100%	100%	100%
1	50	100%	100%	80%	100%
1	20	90%	100%	100%	100%
1	15	90%	100%	100%	100%
1	10	90%	100%	100%	100%
1	5	100%	100%	100%	100%
1	2	100%	100%	100%	100%

complexity is measured by the number of goals and objects in the planning problem. It is easy to see that EVOCK improves drastically with respect to the initial seed (HAMLET seed) by solving very hard problems. The percentage of testing problems solved for PRODIGY working alone, the initial HAMLET seed and the final HAMLET result are also shown.

5 DISCUSSION AND CONCLUSIONS

After having experimented both systems (EVOCK and HAMLET) we can draw the following conclusions and comparisons.

- HAMLET does not have a trade-off between correct knowledge and utility of that knowledge. HAMLET manages to learn quite correct knowledge [Borrajo and Veloso, 1997] but sometimes having a lot of correct control rules is not an advantage, because it takes a long time to use it (this is called the utility problem [Minton, 1988]). This explains in part HAMLET bad behavior. On the other hand, our results in [Aler *et al.*, 1998a] show that it is more difficult for GP alone to obtain correct knowledge. However, it is very easy to take into account the utility problem in the fitness function (several of its components press to that

end). Thus, we see that our multi-strategy approach works better than the two methods alone by combining both methods biases.

- Another problem that HAMLET has is that as it is a lazy incremental system, in order to refine an incorrect control rule it assumes that eventually it will find an appropriate set of negative examples. Given that the potential problem space is infinite (huge from a computational point of view), the likelihood of finding that appropriate set might be very small. In any case, previous work has shown that in the long run HAMLET tends to converge to the correct knowledge [Borrajo and Veloso, 1997]. Since GP a non-incremental system, it is able to detect negative examples at once by evaluating the whole set of training problems. On the other hand, non-incremental methods are less efficient when learning in complex domains. Again, the complementary aspects of both systems allow to overcome both systems deficiencies.
- Another difference between using GP in this way and more traditional learning techniques is that even using background knowledge, its generalization and specialization operators do not have knowledge about how planning acts. On the contrary, learning techniques such as PRODIGY/EBL [Minton, 1988], or HAMLET “know”⁴ how to generalize or specialize in planning domains. GP has no such knowledge, so many of the genetic modifications will not work. Besides, genetic operators are not so constrained by powerful heuristics, so they might get different and new results than those of more traditional methods. Another way to see this is that HAMLET (and many other learning methods) take advantage of the specific-to-general ordering of the control rule space: HAMLET trajectory through the control rule space consists of generalization or specialization steps, in reaction to new examples [Shapiro, 1983]. GP does not take much advantage of this specific-to-general ordering. A mixture of generalizations and specializations are performed at each step in the search. Besides, generalization operators that take advantage of the ordering heuristic are easily added to the operator pool, as our system shows.
- Given that genetic operators do not handle much knowledge, they are faster than classical learning search operators.

⁴Or at least, they have powerful heuristics.

- HAMLET is deterministic: from the same set of training cases, it will always obtain the same set of control rules. On the other hand, GP is stochastic: it can be run several times and obtain different knowledge every time.
- There is a trade-off between understandability and efficiency. HAMLET tends to produce control knowledge which is easier to understand whereas EVOCK control knowledge is more difficult to understand (but more efficient).
- Finally, an important advantage of GP over the rest of learning techniques applied to problem solving is its flexibility. Very different learning biases can be tested without changing the method itself. Following Utgoff's classification [Utgoff, 1986], GP biases are:
 - The language bias can be changed easily. That is not the case with many other learning techniques applied to problem solving, because their search operators depend heavily on the representation language used. For instance, HAMLET only uses a subset of the control rule language allowed by PRODIGY, while GP could use the whole set easily.
 - The exploration bias. GP uses just two task independent operators (crossover and mutation). However, as this paper shows, many possible variations of these operators can be added, as, for instance, task dependent operators (like generalization and specialization).
 - The evaluation bias. In GP, different evaluation biases can be easily combined in the same evaluation function. Also, it is very easy to change from a fitness function to another. In fact, in this paper we have presented a new fitness function that improves previous results obtained using our scheme [Aler *et al.*, 1998a, Aler *et al.*, 1998b].

6 RELATED WORK

There have been different approaches to acquire control knowledge for non-trivial (non-linear) problem solving. Some of them use analogy [Kambhampati, 1989, Veloso and Carbonell, 1993], others pure deduction [Katukam and Kambhampati, 1994, Minton and Zweben, 1993], pure induction [Leckie and Zukerman, 1991], and some combine deduction and induction [Borrajo and Veloso, 1997, Estlin and Mooney, 1996]. The main difference with our approach is that

they did not combine incremental knowledge intensive and non-incremental methods (GP).

Some innovative approaches to problem solving use genetic programming [Koza, 1992]. This approach was started by Koza [Koza, 1989, Koza, 1992], who evolved a planner that solved a very specific set of problems in the blocksworld domain. Handley [Handley, 1994] used GP to evolve plans for specific problems in the blocksworld domain. Muslea [Muslea, 1997] generalized, extended, and formalized this idea, and showed how any planning problem could be translated to an equivalent GP problem. He tested it successfully in several domains. Spector [Spector, 1994] proposed and analyzed several ways in which GP could be used for planning. The main difference with our approach is that they used GP to search in the plans space.

References

- [Aler *et al.*, 1998a] Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Evolving heuristics for planning. In *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, Lecture Notes in Artificial Intelligence, San Diego, CA, March 1998. Springer-Verlag.
- [Aler *et al.*, 1998b] Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Genetic programming of control knowledge for planning. In Reid Simmons, Manuela Veloso, and Stephen Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, Pittsburgh, PA, June 1998.
- [Blum and Furst, 1995] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In Chris S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, pages 1636–1642, Montr al, Canada, August 1995. Morgan Kaufmann.
- [Borrajo and Veloso, 1997] Daniel Borrajo and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 11(1-5):371–405, February 1997.
- [Estlin and Mooney, 1996] Tara A. Estlin and Raymond Mooney. Hybrid learning of search control for partial-order planning. In *New Directions in AI Planning*, pages 115–128. IOS Press, 1996.

- [Fraser and Rush, 1994] Adam P. Fraser and Jon R. Rush. Putting ink into a biro: A discussion of problem domain knowledge for evolutionary robotics. In *AISB Workshop, Evolutionary Computing*, April 11th-13th 1994.
- [Handley, 1994] Simon G. Handley. The automatic generations of plans for a mobile robot via genetic programming with automatically defined functions. In Kenneth E. Kinneer, Jr., editor, *Advances in Genetic Programming*, chapter 18, pages 391-407. MIT Press, 1994.
- [Kambhampati, 1989] Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. PhD thesis, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD, 1989.
- [Katukam and Kambhampati, 1994] Suresh Katukam and Subbarao Kambhampati. Learning explanation-based search control rules for partial order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 582-587, Seattle, WA, 1994. AAAI Press.
- [Koza and Rice, 1991] John R. Koza and James P. Rice. Genetic generation of both the weights and architecture for a neural network. In *Proceedings of IJCNN-91*, volume II, pages 397-404, 1991.
- [Koza, 1989] J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, volume 1, pages 768-774. Morgan Kaufmann, 20-25 August 1989.
- [Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [Leckie and Zukerman, 1991] C. Leckie and I. Zukerman. Learning search control rules for planning: An inductive approach. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 422-426, Evanston, IL, 1991. Morgan Kaufmann.
- [Minton and Zweben, 1993] Steven Minton and Monte Zweben. Learning, planning and scheduling: An overview. In Steven Minton, editor, *Machine Learning Methods for Planning*, chapter 8. Morgan Kaufmann, 1993.
- [Minton, 1988] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.
- [Montana, 1995] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199-230, 1995.
- [Muslea, 1997] Ion Muslea. SINERGY: A linear planner based on genetic programming. In Sam Steel, editor, *Recent Advances in AI Planning. 4th European Conference on Planning, ECP'97*, number LNAI 1348 in Lecture Notes in Artificial Intelligence, pages 312-324, Toulouse, France, September 1997. Springer-Verlag.
- [Penberthy and Weld, 1992] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, pages 103-114, 1992.
- [Shapiro, 1983] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.
- [Spector, 1994] L. Spector. Genetic programming and AI planning systems. In *Proceedings of Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.
- [Utgoff, 1986] Paul Utgoff. *Machine Learning: An Artificial Intelligence Approach*, volume II, chapter Shift of Bias for Inductive Concept Learning, pages 107-148. Morgan Kaufmann, Los Altos, CA, 1986.
- [Veloso and Carbonell, 1993] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249-278, March 1993.
- [Veloso et al., 1995] Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81-120, 1995.

An Experimental Evaluation of Coevolutionary Concept Learning

Cosimo Anglano Attilio Giordana Giuseppe Lo Bello Lorenza Saitta

Dipartimento di Informatica, Università di Torino

Corso Svizzera 185, 10149 Torino, Italy

{mino,attilio,lobello,saitta}@di.unito.it

Abstract

In this paper an extensive experimental evaluation of an evolutionary approach to concept learning is presented. The experimentation, performed with the system G-NET, investigates the effectiveness of the approach along the following dimensions: Robustness with respect to parameter setting, effectiveness of the MDL criterion coupled with a stochastic search bias, impact of coevolution on the quality of the solution and on the computational effort required, and ability to face problems requiring structured representation languages. A discussion of the obtained results and a suggestion on when this type of approach might be useful is also provided.

1 INTRODUCTION

Supervised concept learning has been tackled, so far, with several approaches, including symbolic, connectionist and evolutive ones. Different approaches are better suited to different classes of problems, depending, for instance, on the nature of data or the availability of domain-specific knowledge.

In the hope of making a little step ahead in the direction of matching learning algorithms to problems, in this paper we present an experimental exploration of an evolutionary approach to the task of learning concept descriptions. Our exploration is articulated along three dimensions: The capability of dealing with complex representation languages, such as subsets of predicate logics; the exploitation of distributed architectures, allowing coevolution to be efficiently implemented; the interaction between the stochastic search bias and the Minimum Description Length (MDL)

principle (Rissanen, 1978), used as evaluation criterion of the concept description.

The experimentation has been conducted with a new version of G-NET (Version 2.0), a descendant of the system REGAL (Giordana and Neri, 1996). G-NET's architecture relies on a computational model characterized by the absence of global memory, which extends the diffusion model (Manderik and Spiessens, 1989) previously developed for genetic algorithms. With respect to a previous implementation (Anglano et al., 1997), the version described here includes an explicit coevolutionary strategy based on (Potter et al., 1995), a new objective function based on the MDL principle, and an improved set of genetic operators.

A first point emerged from the experimentation, using both G-NET and REGAL, is that evolutionary search techniques can indeed be fruitfully exploited in concept acquisition. On standard benchmarks they showed performances at least comparable with the best ones presented in the literature (Neri and Saitta, 1996).

A second point is that good performance does not come for free: Using a simple genetic algorithm, easy to understand and quick to implement, may not be a solution. The evolutionary inference engine has to be integrated into a possibly complex architecture, allowing sophisticated description languages, flexible heuristic learning strategies, and distributed computation to be accommodated.

A third point is that evolutionary search proved to be quite robust, because it did not require any parameter tuning over a range of different problems. Finally, stochastic search bias proved to be well suited to different evaluation criteria (Anglano et al., 1997), including the MDL (Rissanen, 1978). G-NET is based, as REGAL was, on the theory of niches and species formation, which already proved to be effective in learn-

ing disjunctive concept definitions (Giordana and Neri, 1996). As niches and species formation is a way of addressing multimodal search problems, disjunctive concept induction naturally fits in this framework. However, methods based on species formation may require large populations when weak species must survive in the presence of much stronger ones (Giordana and Neri, 1996).

In order to cope with this problem, G-NET 2.0 uses a new learning method, which combines the Universal Suffrage selection scheme (Giordana and Neri, 1996) with an explicit coevolutionary strategy, similar to the one proposed in (Potter et al., 1995). Finally, G-NET 2.0 uses a new set of genetic operators, which explicitly aims at preserving the diversity in the population, reducing thus premature convergence and increasing the effectiveness of the genetic search.

2 EVOLUTIONARY ALGORITHM

As its ancestor REGAL, G-NET learns concept descriptions in a language similar to VL_{21} (Michalski, 1983). More specifically, a concept is described by a set $\Phi = \{\varphi_1, \varphi_2, \dots, \varphi_n\}$ of Horn clauses, in which the construct of internal disjunction is also allowed. In Logic Programming, an internal disjunction is a special term describing a set of constants. By setting a limit on the maximum complexity, Horn clauses can be encoded as fixed length bitstrings. A detailed description of the language used by G-NET can be found in (Giordana and Neri, 1996; Giordana et al., 1997).

G-NET's inductive engine exploits a stochastic algorithm organized in two levels. The lower level, named Genetic layer (G-layer), searches for Horn clauses representing partial definitions φ of the concept to learn. The architecture of the G-layer derives from the diffusion model (Manderik and Spiessens, 1989), and integrates different ideas originated inside the field of evolutionary computation and tabu search (Rayward-Smith et al., 1989). The upper level, namely the Supervisor, builds up a global disjunctive definition Φ , out of the partial definitions φ_i 's generated inside the G-layer, using a greedy set covering algorithm. Moreover, the Supervisor interacts with the G-layer according to a coevolutionary strategy (Potter et al., 1995), which aims at increasing the probability of evolving clauses useful to improve the quality of the disjunctive concept description currently in progress.

From a computational point of view, the G-layer consists of a set of elementary searching nodes called G-nodes. Every G-node, G_i , is associated with a single

concept instance ϵ^+ and executes a local evolutionary search aimed at constructing an inductive hypothesis covering ϵ^+ , and having a fitness value as high as possible. The same instance ϵ^+ can be assigned to more than one G-node.

The association between G-nodes and concept instances is dynamically established by the Supervisor, which decides what regions of the hypothesis space to search, and how much.

Every G-node is provided with a small local memory, where it stores the set of current hypotheses it is working on (local population). Basically, the search algorithm executed by a G-node resembles a simple Genetic Algorithm:

G-node Search Algorithm
repeat

1. Select two clauses φ_1 and φ_2 from the local memory with probabilities proportional to their fitness;
2. Create two new clauses φ'_1 and φ'_2 , both different from φ_1 and φ_2 ;
3. Evaluate φ'_1 and φ'_2 on the learning set;
4. Broadcast the new clauses to every G-node associated with some instance ϵ^+ they correctly cover;

until a *halt condition* is reached

The outcome of the evaluation step is a fitness value $f_L(\varphi)$ corresponding to the quality of the clause φ (see below). By generalizing a formula covering the associated instance ϵ^+ , a G-node can implicitly generate formulas also covering other instances which are assigned to different G-nodes. The aim of the broadcasting step is to propagate these formulas to the G-nodes, which potentially can benefit from them. When a clause is broadcast to another G-node, it competes for entering the local memory by playing a kind of stochastic tournament (Harik, 1995), based on the fitness value f_L . As the policy we adopt enforces diversity in the local memories, a clause is allowed to play the tournament only if no copy of it is already there. At the beginning, the population of a G-node is initialized with only one individual and can grow up to a maximum predetermined size. The tournament step is performed only after the maximum size has been reached.

This way of propagating inductive hypotheses among G-nodes promotes the formation of families of hypotheses, which cluster the concept instances into

groups, roughly corresponding to different modalities of the target concept. From the point of view of evolutionary computation this process can be seen as a process of niches and species formation (Goldberg and Richardson, 1987).

The emergence of species (i.e. concept modalities) is the baseline for the coevolutionary strategy adopted by the Supervisor. Periodically, the Supervisor (a) collects the best representatives of each species, and works out a global concept description, (b) reassigns the concept instances to the G-nodes in order to increase the search efforts where emerging species still correspond to low quality inductive hypotheses, and (c) supplies a corrective term to be added to the fitness of the inductive hypotheses in the G-layer, helping the species that better contribute to the global solution to develop further.

3 THE FITNESS EVALUATION

In G-NET, two different fitness functions, f_G and f_L , are used in order to evaluate global (disjunctive) and local (conjunctive) concept descriptions, respectively. Both measures are based on the Minimum Description Length Principle (Rissanen, 1978). The function $f_G(\Phi)$ is the sum of three terms:

$$f_G(\Phi) = MDL_{MAX} - MDL(\epsilon^+(\Phi) + \epsilon^-(\Phi)) - MDL(\Phi) \quad (1)$$

being MDL_{MAX} the MDL of the whole learning set, $MDL(\epsilon^+(\Phi) + \epsilon^-(\Phi))$ the MDL of the set ϵ^+ of positive concept instances not covered by Φ and of the set of negative instances ϵ^- covered by Φ , and $MDL(\Phi)$ the minimum description length of the syntactic form of Φ . In turn, $MDL(\Phi)$ is computed as the sum $MDL(\Phi) = \sum_i MDL(\varphi_i)$ of the MDL of the single clauses belonging to Φ . In all cases, the expressions for the MDL of the different terms have been obtained using Stirling's approximation, as in (Oliveira and Sangiovanni-Vincentelli, 1996). The definition of f_G has been chosen in order to have a function which increases when the MDL decreases, because it is easier to transform it into a probability, used to guide the stochastic search.

The local fitness f_L for evaluating a single clause φ in a G-node takes the form:

$$f_L(\varphi) = MDL_{MAX} - MDL(\varphi) - MDL(\epsilon^-(\varphi)) + (f_G(\Phi') - f_G(\Phi)) \quad (2)$$

being Φ the current global description constructed by the Supervisor, and Φ' the formula obtained by adding

φ to Φ and eliminating all redundant disjuncts but φ . In other words, the second and third term evaluate how simple and consistent φ is. The fourth term is the bias for enforcing the coevolutionary strategy and evaluates how well φ combines with the other existing partial descriptions in order to form a global solution, covering the instance e^+ associated to the G-node and as much as possible of the other instances.

4 THE COEVOLUTION STRATEGY

The Supervisor enforces coevolution by means of two algorithms, which are executed periodically at the end of a *macro-cycle*. A macro-cycle is measured by counting the number of iterations of the Search Algorithm (μ -cycles) globally performed, in the G-layer, by the G-nodes. The first algorithm computes a global concept description Φ out of the best representatives of the species emerged in the G-layer, and is based on a hill climbing optimization strategy. At first, from every G-node the locally best hypothesis is collected and is then merged into a redundant disjunctive description Φ' . Then, Φ' is optimized by eliminating the disjuncts, which are not necessary. This is done by repeating the following cycle until Φ' reaches a final form Φ , which cannot be optimized further:

1. Search the clause φ such that $f_G(\Phi' - \varphi)$ shows the greatest improvement.
2. Set $\Phi' = \Phi' - \varphi$

The second algorithm computes the assignment of the (positive) concept instances to the G-nodes. The basic strategy consists in focusing the search on the concept instances which are covered by poor inductive hypotheses, without omitting to continue the refinement of the other hypotheses. This is done by balancing the computation among the different emerging species, in such a way that species covering smaller niches will get the same computational power as the ones covering larger niches.

The Supervisor keeps track of the solution state of every positive instance $e^+ \in E^+$ (the set of all positive instances), i.e., the best solution found for it. Moreover, it also records the number c_i of μ -cycles, related to e_i^+ , occurred during the past computation. The kernel of the coevolutionary control strategy is the method used for accounting the μ -cycles related to every e_i^+ . As soon as clauses covering many examples will begin to develop, we will find spontaneously born

clusters of G-nodes that elect the same clause as current best hypothesis in their population. This can be interpreted as a form of implicit cooperation, which leads to the generation of a clause, representative of the work of all of them. Therefore, the Supervisor attributes to an instance ϵ_i^+ all the μ -cycles executed by the G-nodes whose local memory contains a copy of the best clause attributed to ϵ_i^+ .

At the end of a macro-cycle, the concept instances are reassigned to G-nodes with the goal of balancing the work spent for every ϵ_i^+ , on the basis of the number c_i of μ -cycles. Let C the maximum value for c_i ($1 \leq i \leq |E^+|$). The Supervisor computes for every ϵ_i^+ the amount $g_i = C - c_i$ of μ -cycles necessary to balance the computational cost for it. Afterwards, the instances are stochastically assigned to G-nodes with probability proportional to g_i . When a G-node G is assigned to a new instance ϵ^+ , it is restarted. If the global description Φ contains a clause φ_e , covering ϵ^+ , φ_e is inserted in the population of G . Otherwise, it will be initialized by means of the seeding operator described below.

5 THE GENETIC OPERATORS

In the same way as REGAL, G-NET represents Horn clauses as fixed length bitstrings ((Giordana et al., 1997)); then, search operators can be implemented as in standard Genetic Algorithms (Goldberg, 1989). As a matter of fact, G-NET uses three basic operators: seeding, crossover and mutation. The seeding operator (Giordana and Neri, 1996) is used for initializing the local memory in the G-nodes when it is empty. When called in a G-node G_i , it stochastically generates a clause φ_i , which is guaranteed to cover the instance ϵ^+ currently associated with G_i .

Crossover and *mutation* operators can be applied in different modalities, depending upon the clauses they are applied to, and are guaranteed to produce new hypotheses different from the parents (original clauses).

The crossover is a combination of the *two point crossover* with a variant of the *uniform crossover* (Syswerda, 1989), modified in order to perform either generalization or specialization of the hypotheses. More specifically, the crossover operator can be activated in three different modalities: exchanging, specializing and generalizing, which are stochastically selected depending on the consistency and completeness of the selected clauses. Given a pair of clauses φ_1, φ_2 , the modality to use is stochastically decided in two steps. In the first step it is decided whether to apply

the exchanging modality, with probability p_{ec} (by default $p_{ec} = 0.1$), or to proceed to the second step, with probability $1 - p_{ec}$. Afterwards, if the second step is entered, the system decides whether to apply generalization or specialization to each one of the parent clauses. Let φ_i be one of the parents; the probability $p_{gc}(\varphi_i)$, of using generalization, and $p_{sc}(\varphi_i) = 1 - p_{gc}(\varphi_i)$, of using specialization, are computed according to the rule:

$$p_{gc}(\varphi_i) = (\epsilon^-(\varphi_i) / (m^+(\varphi_i) + \epsilon^-(\varphi_i))) \quad (3)$$

being m^+ the number of positive instance correctly classified by φ_i and ϵ^- the number of negative instances, as previously defined. Afterwards, if the same modality has been chosen for both operands, the crossover will be applied with this modality. Otherwise, if the modalities are discordant, the exchanging modality will be used.

In this way, the generalizing modality tends to be used when the parents are both consistent, the specializing modality when the parents are both inconsistent, and the exchanging modality when one is consistent and the other is inconsistent. The first decision step guarantees that an assigned percentage of pure information exchange takes place in any case.

In order to guarantee the actual exchange of information, the crossover algorithm first constructs an index $I = \{i_1, i_2, \dots, i_n\}$ of pointers to the positions in the bitstring where the corresponding bits in the two parents have different values. Afterwards, if generalization/specialization has been chosen, two temporary clauses ψ_1 and ψ_2 , identical to φ_1 and φ_2 respectively, are created.

Then, for every element $i_j \in I$ the following procedure is repeated:

- if generalizing modality has been chosen then with probability p_u replace in ψ_1 and ψ_2 the value of the bit $b(i_j)$ with the logical *or* of the corresponding bits in the operands φ_1 and φ_2 .
- if specializing modality has been chosen then with probability p_u replace in ψ_1 and ψ_2 the value of the bit $b(i_j)$ with the logical *and* of the corresponding bits in φ_1 and φ_2 .

If, after applying this stochastic procedure, no bit has been changed, one bit chosen at random in I is generalized/specialized.

When the exchanging modality is chosen, the classical two-point crossover is applied, with the difference that, in order to guarantee an information exchange, the two crossover points are chosen on the index vector I instead of on the whole bitstring.

The *mutation* operator adopts a strategy similar to the one described so far for crossover, and tries to generalize or to specialize an individual, depending on its consistency or inconsistency. Also the mutation operator can have three modalities, namely seeding, generalizing or specializing, which are selected with probability p_{seed} (by default $p_{seed} = 0.1$), p_{gm} and p_{sm} , respectively. The probabilities p_{gm} for generalizing mutation and p_{sm} for specializing mutation are computed with the rule:

$$\begin{aligned} p_{sm} &= (1 - p_{seed})(\epsilon^- / (\epsilon^- + m^+)), \\ p_{gm} &= 1 - p_{seed} - p_{sm}, \end{aligned} \quad (4)$$

where the argument of ϵ^- and m^+ has been omitted for brevity. If the specializing mutation is chosen, the mutation is applied as follows: let n_1 be the number of bits set to "1" in the bitstring; then, the mutation operator turns to "0" a fraction γ of them, which is obtained by randomly selecting a real number in the interval $[0, n_1/10]$. The bits to be set to "1" are selected in an analogous way, when the generalizing mutation is chosen.

It is easy to recognize that generalizing and specializing mutations are nothing else than the dropping condition and adding condition operators defined in (Jong et al., 1993).

In the cycle executed by each G-node, two clauses are selected at each iteration with probability proportional to their fitness f_L . If the population is empty, a new individual will be created using the seeding operator. Otherwise, if the two selected clauses φ_1 and φ_2 are different, crossover is applied. On the contrary, if the same clause is selected two times, two new clauses are created using mutation.

The nice aspect of this strategy is that it automatically adapts to the composition of the population. When the population in a node is dominated by a clause that has a fitness much higher than the others (and, then, it is frequently selected for reproduction with itself), the search turns into a stochastic hill climbing.

6 EXPERIMENTAL EVALUATION

In the following we present an extensive evaluation of G-NET made on a variety of datasets, selected

with the aim of testing the system with respect to the dimensions mentioned in Section 1: language bias, robustness to evaluation criteria, and overall performance. The parameters to tune are actually very few (the genetic operators constants are not user tunable) and correspond to the local population size P_s , the macro-cycle size M_c and the number of G-nodes N_g . In all the previous experimentation done, they did not appeared to be critical at all and the following setting has been chosen as a default: $P_s = 10$, $M_c = 300$, $N_g = 100$. The results reported in the following have been obtained using the default setting.

Table 1 reports a first group of results on datasets used to test the system Smog (Oliveira and Sangiovanni-Vincentelli, 1996), which exploits the MDL as hypothesis evaluation criterion. Results by C4.5 are used as a baseline. The performance for Smog and C4.5 are those reported in (Oliveira and Sangiovanni-Vincentelli, 1996); Smog used many other datasets, but only some of them are available at the U.C. Irvine repository (Merz et al., 1991).

G-NET has always been run with a set of 100 G-nodes and has been stopped after creating 40000 hypotheses. The specific goal of the test was twofold: to investigate how G-NET is affected by changing its evaluation criterion, all the rest being the same, and whether the MDL could still be effective when coupled with a stochastic search bias, such as the one provided by G-NET, very different from the ones used in Smog and in C4.5. The answer has been positive in both cases.

By considering the results on the *Monk-2* dataset, the effectiveness of G-NET's species formation mechanism is evident: the system always found 26 disjuncts, sometimes the correct ones and sometimes little different; this explains the small error of the acquired knowledge base. The species formation stability is also confirmed by the fact that in all cases G-NET found the same number of disjuncts, differing for small variations.

Table 2 reports results of an experiment aimed at verifying the utility of increasing the computational power of the search when approaching a more large and difficult problem. The dataset used is the *Splice Junctions* dataset (Towell and Shavlik, 1994). The task is that of identifying boundaries between coding (exons) and non-coding (introns) regions of genes occurring in eukaryote DNA.

The Splice Junctions dataset has been previously used to test the system REGAL, which presented the best results so far among the many reported in the literature (Neri and Saitta, 1996). While increasing the

Table 1: Comparison Between G-NET, Smog And C4.5 With Respect To The Average Error Rate Of The Solution, Evaluated With The 10-fold Crossvalidation

Problem	Dataset size	Average Error %			Average N. of Disjuncts
		G-NET	Smog	C4.5	G-NET
monk1	432 10-fold	0.00 \pm 0.00	0.00 \pm 0.00	0.00 \pm 0.00	3.0
monk2	432 10-fold	2.80 \pm 3.80	0.00 \pm 0.00	32.83 \pm 10.66	26.0
monk3	432 10-fold	0.00 \pm 0.00	0.00 \pm 0.00	0.00 \pm 0.00	3.0
tictactoe	958 10-fold	0.97 \pm 0.62	2.82 \pm 1.97	7.07 \pm 1.82	10.5
credit	690 10-fold	15.8 \pm 4.40	19.57 \pm 5.08	14.03 \pm 3.28	14.0
breast	699 10-fold	5.29 \pm 2.89	6.72 \pm 2.44	5.85 \pm 3.32	2.6
vote	435 10-fold	5.10 \pm 3.20	5.29 \pm 2.64	4.63 \pm 3.05	2.0

Table 2: Comparison Between G-NET And REGAL With Respect To The Average Error Rate And The Complexity Of The Solution

Problem	Dataset size	Average Error %		N. of Disjuncts	
		G-NET	REGAL	G-NET	REGAL
splice-j. (EI)	2000 + 1190	3.40	4.40	7	19
splice-j. (IE)	2000 + 1190	2.90	4.20	10	26
splice-j. (Neither)	2000 + 1190	3.30	5.20	11	21
mushrooms	4000 + 4124	0.00	0.00	3	6

system parallelism and decreasing the complexity, G-NET achieved even lower average error rate (error rate is an average over 3 runs). The second best results were achieved by KBANN (Towell and Shavlik, 1994): 7.56% for EI, 8.47% for IE, and 4.62% for Neither. This comparison suggests that genetic search could be better suited to complex problems.

Finally, Table 3 reports the results of experiments aimed at confirming that G-NET (as its predecessor REGAL) is able to effectively deal with more complex languages, such as predicate logic based ones.

The first row in Table 3 refers to the *mutagensis* dataset, a challenging problem widely used in the ILP community for testing induction algorithms in First Order Logic (King et al., 1995). The problem consists in learning rules for discriminating substances having cancerogenetic properties on the basis of their chemical structure. The difficulty lies mainly in the complexity of matching formulas in First Order Logic, which limits the exploration capabilities of any induction system. To our knowledge, the best results with this database have been obtained by STILL (Sebag, 1997) a stochastic induction algorithm that easily reaches error rates below 10%, and, with a careful setting of the control

parameters, made the best hit at 6.4%. Many other systems, going from Linear Regression to PROGOL and FOIL, reported error rates ranging between 11% and 14%. G-NET, using only the predicates used in (Sebag, 1997), obtained an error rate of 8.8%.

The second case study is a classification problem (Esposito et al., 1992) of documents acquired through a scanner, and processed by an image processing program that produces a structured description of the layout. The dataset contains structured data described with 5 symbolic and 3 numeric attributes, and has been used to test learners with the capability of dealing with numerical features in FOL (Esposito et al., 1992; Botta and Giordana, 1993). G-NET does not have, at the moment, any specific strategy for dealing with numerical features, and so we transformed the problem into a symbolic one by discretizing the numeric features. Each numeric feature has been discretized by subdividing the range into 16 equal length intervals. G-NET easily reached an error rate below the 1%, approximately the same as SMART+ which has specific strategies for dealing with numerical features.

Finally, the last case study (*Train Checkout-3*) is a

Table 3: Experiments With First Order Problems. Error Rate For The *Train Check-out 3* Is An Average Of 3 Runs

Problem	Dataset size	Average Error %			N. of Disjuncts
		G-NET	STILL	SMART + FONNs	
mutagenesis	230 10-fold	8.80 ± 7.90	6.4 ± 4.5	n.a.	3
office-doc	210 + 160	0.89 ± 0.72	n.a.	0.80	11
train check-out 3	500 + 6000	11.3 ± 0.47	n.a.	16.8	2

difficult artificial dataset generated for testing FONNs (Botta et al., 1997), a kind of neural network recently proposed for refining numerical terms in Horn Clauses. The dataset contains the description of a set of trains, similar to the one proposed by Michalski, where each coach is described by means of a set of 5 symbolic and 4 numerical attributes. In (Botta et al., 1997) three different learning problems of increasing difficulty are presented, related to this dataset. The problems consist in learning sets of rules for assessing when a train meets the safety conditions required for travelling on a given line. The one we considered here is the most difficult among them and the challenge is to discover the rule used for classifying the concept instances:

a train cannot go if it contains two near cars, both without brakes and heavier than a threshold we_3 or if it contains two near cars carrying an unstable load (special material) and heavier than a threshold $we_4 < we_3$.

FONNs could easily reach an error rate below 2% on a test set of 6000 instances starting from a handcrafted knowledge base, which correctly described the structure of the rule hidden in the data, but only reached an error rate of about 17% starting from a set of rules learned by SMART+ from 500 learning instances. Reshaping the problem in propositional calculus, C4.5 and CART could not go below an error rate of 27%, and neural networks such as multi-layer perceptron and cascade correlation were performing even more poorly (Botta et al., 1997).

G-NET has been run by discretizing every numeric attribute into a range of 30 intervals. As it appears from the last row in Table 3, it was able to find two clauses which show an error rate around 11%.

7 DISCUSSION

As it appears from the results reported above, G-NET is a very flexible system, able to deal with many different problems, producing good results. Moreover, as already stated, the results have been obtained without performing any specific tuning, so that the system proved to be quite robust and easy to use. This looks surprising considering that a major complaint against GAs is the difficulty of tuning parameters.

We point out that, in spite of its architecture strongly resembling a Genetic Algorithm, G-NET cannot be considered a classical GA, because the principles which control the evolution are substantially different. In our opinion, two aspects determine the success of G-NET: the enforcement of diversity in the local populations and the coevolution.

In their basic formulation GAs use genetic pressure, i.e. the capability of the most fit individuals to reproduce more quickly, so that the weakest ones are eliminated from the population. This mechanism has the positive effect of focusing the search on the most fit individuals, so that, in the best case, the algorithm will climb up a maximum of the fitness function. Unfortunately, the mechanism is unstable and a too quick convergence prevents reaching optimal solutions. Another drawback is that, in this way, many identical individuals will be present in the population, so that the search can become ineffective because the major search operator (crossover) reproduces again and again the same individuals.

A trend in the GA literature, which at least partially relieves this problem, is related to the theory of species and niches formation. Species formation can be promoted in many ways by limiting the genetic pressure between species (Goldberg and Richardson, 1987). Species formation offers some benefits, such as the possibility of restricting crossover to the individuals of the same species (crossover among dif-

ferent species is essentially deceptive), increasing the search effectiveness and allowing the discovery of multiple modalities. For instance, in G-NET, as well as in REGAL, this has been exploited for learning disjunctive concept descriptions. However, even in this framework, genetic pressure continues to be used inside a same species as a mechanism of focus of attention. Requiring that a population (the local memory of G-nodes) contains individuals (clauses) all different, is a definite departure from this mechanism, and drastically limits any form of genetic pressure. Therefore, the algorithm becomes much more stable and less sensitive of crossover type, and of crossover and mutation rates. Furthermore, in the place of genetic pressure other strategies, tailored to the specific task, can be used for guiding the search. In our case, the coevolution is now the major strategy that focuses the search where it is necessary instead of letting it follow the stream enforced by the genetic pressure. A second component is represented by the local search operators, which are context sensitive and make the best effort in order to increase the exploration capability of the algorithm.

Both the idea of maintaining the population diversity and the one of coevolution originated before G-NET, whose originality consists in the adaptation to the specific task and to the integration of these ideas into a unique framework. On the one hand, diversity in GAs has been already proposed by several authors (Augier et al., 1995), although no one speculates on the reasons why a GA should benefit from it. On the other hand, diversity could be related to tabu search. The local memory of a G-nodes works as an elementary tabu list which prevents the algorithm from reprocessing already generated instances without an explicit will to do so.

Coevolution appeared inside the GA community several years ago (Husbands and Mill, 1991), and has been considered by few others in the following. The coevolution model, described here, conforms to the one proposed by (Potter et al., 1995), properly re-interpreted in the framework of concept learning, which naturally conforms to it.

Finally, the reassignment of the examples to be covered to G-nodes, performed by the Supervisor, can be considered a kind of boosting (Shapire, 1990): in subsequent runs, the search efforts shall be concentrated on those parts of the hypothesis space not yet adequately covered. Currently, the series of found hypotheses are combined into a unique formula, which differentiate this approach from a genuine boosting.

However, nothing hinders the Supervisor from keeping apart the hypotheses and using them according to a majority voting classification strategy, instead of combining them. This possibility has not been explored yet.

8 CONCLUSIONS

In this paper we presented a new induction system based on an evolutionary approach, which is the outcome of several years of investigation in this direction.

Given the good results obtained across a variety of datasets, languages, and evaluation criteria, it should be evident that a system like G-NET can be profitably used to explore the structure of new learning problems, when little a priori information, clearly pointing to another approach, is available.

Moreover, thanks to its computational model, G-NET is able to effectively exploit parallel computing systems, allowing to deal with large and complex datasets. As a matter of fact, in addition to the possibility of distributing the search among many G-nodes, G-NET offers also the possibility of distributing the hypotheses evaluation on several processors. Although this aspect has not been described here, because it is outside the scope of the paper, the current implementation of G-NET runs on a cluster of workstations (Anglano et al., 1997). This facility has been extensively exploited for the experiments on *Mutagenesis* and *Splice Junctions* datasets, so that the results for every run have been obtained in a few hours.

The conclusion is that G-NET seems to be very well-suited to learning structured concepts, such as the ones typically learned by ILP methods, and, in addition, to face learning problems on large databases.

References

- Anglano, C., Giordana, A., Lo Bello, G., and Saitta, L. (1997). A network genetic algorithm for concept learning. In *Int. Conf. on Genetic Algorithms*, pages 434–441, Lansing, MI. Morgan Kaufmann.
- Augier, S., Venturini, G., and Kodratoff, Y. (1995). Learning first order rules with a genetic algorithm. In *Proc. of the First International Conference On Knowledge Discovery and Data mining*, pages 21–26, Montreal, Quebec, CA. AAAI Press.
- Botta, M. and Giordana, A. (1993). SMART+: A multi-strategy learning tool. In *IJCAI-93*,

- Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, volume 2, Chambéry, France.
- Botta, M., Giordana, A., and Piola, R. (1997). Fonn: Combining first order logic with connectionist learning. In *14-th International Conference on Machine learning ICML-97*, pages 157–166, Nashville, TN. Morgan Kaufmann.
- Esposito, F., Malerba, D., and Semeraro, G. (1992). Classification in noisy environments using a distance measure between structural symbolic descriptions. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 14(3):390–402.
- Giordana, A. and Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation*, 3:375–416.
- Giordana, A., Neri, F., Saitta, L., and Botta, M. (1997). Integrating multiple learning strategies in first order logics. *Machine Learning*, pages 209–240.
- Goldberg, D. (1989). *Genetic Algorithms*. Addison-Wesley.
- Goldberg, D. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Int. Conf. on Genetic Algorithms*, pages 41–49, Cambridge, MA. Morgan Kaufmann.
- Harik, G. (1995). Finding multimodal solutions using restricted tournament selection. In *Int. Conf. on Genetic Algorithms*, pages 24–31, Pittsburgh, PA. Morgan Kaufmann.
- Husbands, P. and Mill, F. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. In *4th Int. Conf. on Genetic Algorithms*, pages 264–270, Fairfax, VA. Morgan Kaufmann.
- Jong, K. D., Spears, W., and Gordon, F. (1993). Using genetic algorithms for concept learning. *Machine Learning Journal*, 13, pages 161–188.
- King, R., Srinivasan, A., and Stenberg, M. (1995). Relating chemical activity to structure: an examination of ilp successes. *New Generation Computing*, 13.
- Manderik, B. and Spiessens, P. (1989). Fine-grained parallel genetic algorithms. In *Int. Conf. on Genetic Algorithms*, pages 428–433, Fairfax, VA. Morgan Kaufmann.
- Merz, C., Murphy, P., and Aha, D. (1991). *Repository of Machine Learning Databases*. University of California, Irvine, CA.
- Michalski, R. (1983). A theory and methodology of inductive learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134, Los Altos, CA. Morgan Kaufmann.
- Neri, F. and Saitta, L. (1996). Genetic algorithms for pattern recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, PAMI-18:1135–1142.
- Oliveira, A. and Sangiovanni-Vincentelli, A. (1996). Using the minimum description length principle to infer reduced ordered decision graphs. *Machine Learning*, pages 23–50.
- Potter, M., DeJong, K., and Grefenstette, J. (1995). A coevolutionary approach to learning sequential decision rules. In *Int. Conf. on Genetic Algorithms*, pages 366–372, Pittsburgh, PA. Morgan Kaufmann.
- Rayward-Smith, V., Osman, I., Reeves, C., and Smith, G. (1989). *Modern Heuristic Search Methods*. Wiley, New York, NY.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.
- Sebag, M. (1997). Tractable induction and classification in first order logic. In *15th International Joint Conference on Artificial Intelligence*, Tokyo, Japan.
- Shapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5:197–227.
- Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *3th Int. Conf. on Genetic Algorithms*, pages 2–9, Fairfax, VA. Morgan Kaufmann.
- Towell, G. and Shavlik, J. (1994). Knowledge based artificial neural networks. *Artificial Intelligence*, 70(4):119–166.

KnightCap: A chess program that learns by combining TD(λ) with game-tree search

Jonathan Baxter

Department of Systems Engineering
Australian National University
Canberra 0200, Australia
Jon.Baxter@anu.edu.au

Andrew Tridgell

Department of Computer Science
Australian National University
Canberra 0200, Australia
Andrew.Tridgell@anu.edu.au

Lex Weaver

Department of Computer Science
Australian National University
Canberra 0200, Australia
Lex.Weaver@anu.edu.au

Abstract

In this paper we present TDLeaf(λ), a variation on the TD(λ) algorithm that enables it to be used in conjunction with game-tree search. We present some experiments in which our chess program “KnightCap” used TDLeaf(λ) to learn its evaluation function while playing on the Free Internet Chess Server (FICS, fics.onenet.net). The main success we report is that KnightCap improved from a 1650 rating to a 2150 rating in just 308 games and 3 days of play. As a reference, a rating of 1650 corresponds to about level B human play (on a scale from E (1000) to A (1800)), while 2150 is human master level. We discuss some of the reasons for this success, principle among them being the use of on-line, rather than self-play.

1 Introduction

Temporal Difference learning, first introduced by Samuel [5] and later extended and formalized by Sutton [7] in his TD(λ) algorithm, is an elegant technique for approximating the expected long term future cost (or *cost-to-go*) of a stochastic dynamical system as a function of the current state. The mapping from states to future cost is implemented by a parameterized function approximator such as a neural network. The parameters are updated online after each state transition, or possibly in batch updates after several state transitions. The goal of the algorithm is to improve the cost estimates as the number of observed state transitions and associated costs increases.

Perhaps the most remarkable success of TD(λ) is Tesauro’s TD-Gammon, a neural network backgammon player that was trained from scratch using TD(λ) and simulated self-play. TD-Gammon is competitive with the best human

backgammon players [9]. In TD-Gammon the neural network played a dual role, both as a predictor of the expected cost-to-go of the position and as a means to select moves. In any position the next move was chosen greedily by evaluating all positions reachable from the current state, and then selecting the move leading to the position with smallest expected cost. The parameters of the neural network were updated according to the TD(λ) algorithm after each game.

Although the results with backgammon are quite striking, there is lingering disappointment that despite several attempts, they have not been repeated for other board games such as othello, Go and the “drosophila of AI” — chess [10, 12, 6].

Many authors have discussed the peculiarities of backgammon that make it particularly suitable for Temporal Difference learning with self-play [8, 6, 4]. Principle among these are *speed of play*: TD-Gammon learnt from several hundred thousand games of self-play, *representation smoothness*: the evaluation of a backgammon position is a reasonably smooth function of the position (viewed, say, as a vector of piece counts), making it easier to find a good neural network approximation, and *stochasticity*: backgammon is a random game which forces at least a minimal amount of exploration of search space.

As TD-Gammon in its original form only searched one-ply ahead, we feel this list should be appended with: *shallow search is good enough against humans*. There are two possible reasons for this; either one does not gain a lot by searching deeper in backgammon (questionable given that recent versions of TD-Gammon search to three-ply and this significantly improves their performance), or humans are simply incapable of searching deeply and so TD-Gammon is only competing in a pool of shallow searchers. Although we know of no psychological studies investigating the depth to which humans search in backgammon, it is plausible that the combination of high branching fac-

tor and random move generation makes it quite difficult to search more than one or two-ply ahead. In particular, random move generation effectively prevents selective search or “forward pruning” because it enforces a lower bound on the branching factor at each move.

In contrast, finding a representation for chess, othello or Go which allows a small neural network to order moves at one-ply with near human performance is a far more difficult task. It seems that for these games, reliable tactical evaluation is difficult to achieve without deep lookahead. As deep lookahead invariably involves some kind of minimax search, which in turn requires an exponential increase in the number of positions evaluated as the search depth increases, the computational cost of the evaluation function has to be low, ruling out the use of expensive evaluation functions such as neural networks. Consequently most chess and othello programs use linear evaluation functions (the branching factor in Go makes minimax search to any significant depth nearly infeasible).

In this paper we introduce TDLeaf(λ), a variation on the TD(λ) algorithm that can be used to learn an evaluation function for use in deep minimax search. TDLeaf(λ) is identical to TD(λ) except that instead of operating on the positions that occur during the game, it operates on the leaf nodes of the *principal variation* of a minimax search from each position (also known as the *principal leaves*).

To test the effectiveness of TDLeaf(λ), we incorporated it into our own chess program—*KnightCap*. KnightCap has a particularly rich board representation enabling relatively fast computation of sophisticated positional features, although this is achieved at some cost in speed (KnightCap is about 10 times slower than *Crafty*—the best public-domain chess program—and 6,000 times slower than *Deep Blue*). We trained KnightCap’s linear evaluation function using TDLeaf(λ) by playing it on the Free Internet Chess Server (FICS, fics.onenet.net) and on the Internet Chess Club (ICC, chessclub.com). Internet play was used to avoid the premature convergence difficulties associated self-play¹. The main success story we report is that starting from an evaluation function in which all coefficients were set to zero except the values of the pieces, KnightCap went from a 1650-rated player to a 2150-rated player in just three days and 308 games. KnightCap is an ongoing project with new features being added to its evaluation function all the time. We use TDLeaf(λ) and Internet play to tune the coefficients of these features.

¹Randomizing move choice is another way of avoiding problems associated with self-play (this approach has been tried in Go [6]), but the advantage of the Internet is that more information is provided by the opponents play.

The remainder of this paper is organized as follows. In section 2 we describe the TD(λ) algorithm as it applies to games. The TDLeaf(λ) algorithm is described in section 3. Experimental results for internet-play with KnightCap are given in section 4. Section 5 contains some discussion and concluding remarks.

2 The TD(λ) algorithm applied to games

In this section we describe the TD(λ) algorithm as it applies to playing board games. We discuss the algorithm from the point of view of an *agent* playing the game.

Let S denote the set of all possible board positions in the game. Play proceeds in a series of moves at discrete time steps $t = 1, 2, \dots$. At time t the agent finds itself in some position $x_t \in S$, and has available a set of moves, or *actions* A_{x_t} (the legal moves in position x_t). The agent chooses an action $a \in A_{x_t}$ and makes a transition to state x_{t+1} with probability $p(x_t, x_{t+1}, a)$. Here x_{t+1} is the position of the board after the agent’s move and the opponent’s response. When the game is over, the agent receives a scalar reward, typically “1” for a win, “0” for a draw and “-1” for a loss.

For ease of notation we will assume all games have a fixed length of N (this is not essential). Let $r(x_N)$ denote the reward received at the end of the game. If we assume that the agent chooses its actions according to some function $a(x)$ of the current state x (so that $a(x) \in A_x$), the expected reward from each state $x \in S$ is given by

$$J^*(x) := E_{x_N|x} r(x_N), \quad (1)$$

where the expectation is with respect to the transition probabilities $p(x_t, x_{t+1}, a(x_t))$ and possibly also with respect to the actions $a(x_t)$ if the agent chooses its actions stochastically.

For very large state spaces S it is not possible store the value of $J^*(x)$ for every $x \in S$, so instead we might try to approximate J^* using a parameterized function class $\tilde{J}: S \times \mathbb{R}^k \rightarrow \mathbb{R}$, for example linear function, splines, neural networks, etc. $\tilde{J}(\cdot, w)$ is assumed to be a differentiable function of its parameters $w = (w_1, \dots, w_k)$. The aim is to find a parameter vector $w \in \mathbb{R}^k$ that minimizes some measure of error between the approximation $\tilde{J}(\cdot, w)$ and $J^*(\cdot)$. The TD(λ) algorithm, which we describe now, is designed to do exactly that.

Suppose x_1, \dots, x_{N-1}, x_N is a sequence of states in one game. For a given parameter vector w , define the *temporal difference* associated with the transition $x_t \rightarrow x_{t+1}$ by

$$d_t := \tilde{J}(x_{t+1}, w) - \tilde{J}(x_t, w). \quad (2)$$

Note that d_t measures the difference between the reward predicted by $\tilde{J}(\cdot, w)$ at time $t + 1$, and the reward predicted by $\tilde{J}(\cdot, w)$ at time t . The true evaluation function J^* has the property

$$E_{x_{t+1}|x_t} [J^*(x_{t+1}) - J^*(x_t)] = 0,$$

so if $\tilde{J}(\cdot, w)$ is a good approximation to J^* , $E_{x_{t+1}|x_t} d_t$ should be close to zero. For ease of notation we will assume that $\tilde{J}(x_N, w) = r(x_N)$ always, so that the final temporal difference satisfies

$$d_{N-1} = \tilde{J}(x_N, w) - \tilde{J}(x_{N-1}, w) = r(x_N) - \tilde{J}(x_{N-1}, w).$$

That is, d_{N-1} is the difference between the true outcome of the game and the prediction at the penultimate move.

At the end of the game, the TD(λ) algorithm updates the parameter vector w according to the formula

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right] \quad (3)$$

where $\nabla \tilde{J}(\cdot, w)$ is the vector of partial derivatives of \tilde{J} with respect to its parameters. The positive parameter α controls the learning rate and would typically be “annealed” towards zero during the course of a long series of games. The parameter $\lambda \in [0, 1]$ controls the extent to which temporal differences propagate backwards in time. To see this, compare equation (3) for $\lambda = 0$:

$$\begin{aligned} w &:= w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) d_t \\ &= w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) [\tilde{J}(x_{t+1}, w) - \tilde{J}(x_t, w)] \end{aligned} \quad (4)$$

and $\lambda = 1$:

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) [r(x_N) - \tilde{J}(x_t, w)]. \quad (5)$$

Consider each term contributing to the sums in equations (4) and (5). For $\lambda = 0$ the parameter vector is being adjusted in such a way as to move $\tilde{J}(x_t, w)$ —the predicted reward at time t —closer to $\tilde{J}(x_{t+1}, w)$ —the predicted reward at time $t + 1$. In contrast, TD(1) adjusts the parameter vector in such a way as to move the predicted reward at time step t closer to the final reward at time step N . Values of λ between zero and one interpolate between these two behaviors. Note that (5) is equivalent to gradient descent on the error function $E(w) := \sum_{t=1}^{N-1} [r(x_N) - \tilde{J}(x_t, w)]^2$.

Successive parameter updates according to the TD(λ) algorithm should, over time, lead to improved predictions of the expected reward $\tilde{J}(\cdot, w)$. Provided the actions $a(x_t)$ are independent of the parameter vector w , it can be shown that for *linear* $\tilde{J}(\cdot, w)$, the TD(λ) algorithm converges to a near-optimal parameter vector [11]. Unfortunately, there is no such guarantee if $\tilde{J}(\cdot, w)$ is non-linear [11], or if $a(x_t)$ depends on w [2].

3 Minimax Search and TD(λ)

For argument’s sake, assume any action a taken in state x leads to predetermined state which we will denote by x'_a . Once an approximation $\tilde{J}(\cdot, w)$ to J^* has been found, we can use it to choose actions in state x by picking the action $a \in A_x$ whose successor state x'_a minimizes the opponent’s expected reward²:

$$a^*(x) := \operatorname{argmin}_{a \in A_x} \tilde{J}(x'_a, w). \quad (6)$$

This was the strategy used in TD-Gammon. Unfortunately, for games like othello and chess it is very difficult to accurately evaluate a position by looking only one move or *ply* ahead. Most programs for these games employ some form of *minimax* search. In minimax search, one builds a tree from position x by examining all possible moves for the computer in that position, then all possible moves for the opponent, and then all possible moves for the computer and so on to some predetermined depth d . The leaf nodes of the tree are then evaluated using a heuristic evaluation function (such as $\tilde{J}(\cdot, w)$), and the resulting scores are propagated back up the tree by choosing at each stage the move which leads to the best position for the player on the move. See figure 1 for an example game tree and its minimax evaluation. With reference to the figure, note that the evaluation assigned to the root node is the evaluation of the leaf node of the *principal variation*; the sequence of moves taken from the root to the leaf if each side chooses the best available move.

In practice many engineering tricks are used to improve the performance of the minimax algorithm, $\alpha - \beta$ search being the most famous.

Let $\tilde{J}_d(x, w)$ denote the evaluation obtained for state x by applying $\tilde{J}(\cdot, w)$ to the leaf nodes of a depth d minimax search from x . Our aim is to find a parameter vector w such that $\tilde{J}_d(\cdot, w)$ is a good approximation to the expected reward J^* . One way to achieve this is to apply the TD(λ) algorithm to $\tilde{J}_d(x, w)$. That is, for each sequence of posi-

²If successor states are only determined stochastically by the choice of a , we would choose the action minimizing the expected reward over the choice of successor states.

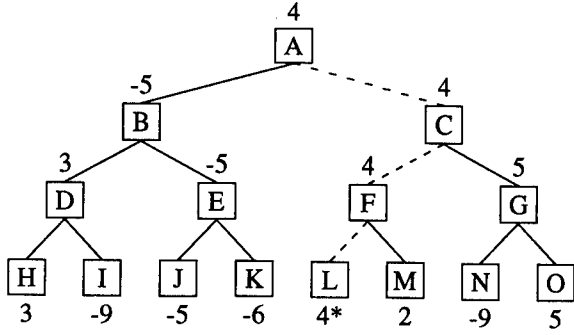


Figure 1: Full breadth, 3-ply search tree illustrating the minimax rule for propagating values. Each of the leaf nodes (H–O) is given a score by the evaluation function, $\tilde{J}(\cdot, w)$. These scores are then propagated back up the tree by assigning to each opponent's internal node the minimum of its children's values, and to each of our internal nodes the maximum of its children's values. The principle variation is then the sequence of best moves for either side starting from the root node, and this is illustrated by a dashed line in the figure. Note that the score at the root node A is the evaluation of the leaf node (L) of the principal variation. As there are no ties between any siblings, the derivative of A's score with respect to the parameters w is just $\nabla \tilde{J}(L, w)$.

tions x_1, \dots, x_N in a game we define the temporal differences

$$d_t := \tilde{J}_d(x_{t+1}, w) - \tilde{J}_d(x_t, w) \quad (7)$$

as per equation (2), and then the TD(λ) algorithm (3) for updating the parameter vector w becomes

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]. \quad (8)$$

One problem with equation (8) is that for $d > 1$, $\tilde{J}_d(x, w)$ is not a necessarily a differentiable function of w for all values of w , even if $\tilde{J}(\cdot, w)$ is everywhere differentiable. This is because for some values of w there will be “ties” in the minimax search, i.e. there will be more than one best move available in some of the positions along the principal variation, which means that the principal variation will not be unique (see figure 2). Thus, the evaluation assigned to the root node, $\tilde{J}_d(x, w)$, will be the evaluation of any one of a number of leaf nodes.

Fortunately, under some mild technical assumptions on the behavior of $\tilde{J}(x, w)$, it can be shown that for each state x , the set of $w \in \mathbb{R}^k$ for which $\tilde{J}_d(x, w)$ is not differentiable has Lebesgue measure zero. Thus for all states x and for “almost all” $w \in \mathbb{R}^k$, $\tilde{J}_d(x, w)$ is a differentiable function

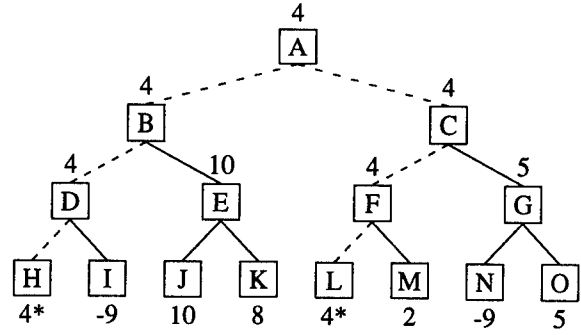


Figure 2: A search tree with a non-unique principal variation (PV). In this case the derivative of the root node A with respect to the parameters of the leaf-node evaluation function is multi-valued, either $\nabla \tilde{J}(H, w)$ or $\nabla \tilde{J}(L, w)$. Except for transpositions (in which case H and L are identical and the derivative is single-valued anyway), such “collisions” are likely to be extremely rare, so in TDLeaf(λ) we ignore them by choosing a leaf node arbitrarily from the available candidates.

of w . Note that $\tilde{J}_d(x, w)$ is also a continuous function of w whenever $\tilde{J}(x, w)$ is a continuous function of w . This implies that even for the “bad” pairs (x, w) , $\nabla \tilde{J}_d(x, w)$ is only undefined because it is multi-valued. Thus we can still arbitrarily choose a particular value for $\nabla \tilde{J}_d(x, w)$ if w happens to land on one of the bad points;

Based on these observations we modified the TD(λ) algorithm to take account of minimax search in an almost trivial way: instead of working with the root positions x_1, \dots, x_N , the TD(λ) algorithm is applied to the leaf positions found by minimax search from the root positions. We call this algorithm TDLeaf(λ). Full details are given in figure 3.

4 TDLeaf(λ) and Chess

In this section we describe the outcome of several experiments in which the TDLeaf(λ) algorithm was used to train the weights of a linear evaluation function in our chess program “KnightCap”. KnightCap is a reasonably sophisticated computer chess program for Unix systems. It has all the standard algorithmic features that modern chess programs tend to have as well as a number of features that are much less common. For more details on KnightCap, including the source code, see wwwsyseng.anu.edu.au/lsg.

Let $\tilde{J}(\cdot, w)$ be a class of evaluation functions parameterized by $w \in \mathbb{R}^k$. Let x_1, \dots, x_N be N positions that occurred during the course of a game, with $r(x_N)$ the outcome of the game. For notational convenience set $\tilde{J}(x_N, w) := r(x_N)$.

1. For each state x_i , compute $\tilde{J}_d(x_i, w)$ by performing minimax search to depth d from x_i and using $\tilde{J}(\cdot, w)$ to score the leaf nodes. Note that d may vary from position to position.
2. Let x_i^l denote the leaf node of the principle variation starting at x_i . If there is more than one principal variation, choose a leaf node from the available candidates at random. Note that

$$\tilde{J}_d(x_i, w) = \tilde{J}(x_i^l, w). \quad (9)$$

3. For $t = 1, \dots, N - 1$, compute the temporal differences:

$$d_t := \tilde{J}(x_{t+1}^l, w) - \tilde{J}(x_t^l, w). \quad (10)$$

4. Update w according to the TDLeaf(λ) formula:

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t^l, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]. \quad (11)$$

Figure 3: The TDLeaf(λ) algorithm

4.1 Experiments with KnightCap

In our main experiment we took KnightCap's evaluation function and set all but the material parameters to zero. The material parameters were initialized to the standard "computer" values: 1 for a pawn, 4 for a knight, 4 for a bishop, 6 for a rook and 12 for a queen. With these parameter settings KnightCap (under the pseudonym "Wimp-Knight") was started on the Free Internet Chess server (FICS, fics.onenet.net) against both human and computer opponents. We played KnightCap for 25 games without modifying its evaluation function so as to get a reasonable idea of its rating. After 25 games it had a blitz (fast time control) rating of 1650 ± 50^3 , which put it at about B-grade human performance (on a scale from E (1000) to A (1800)), although of course the kind of game KnightCap plays with just material parameters set is very different to human play of the same level (KnightCap makes no short-term tactical errors but is positionally completely ignorant). We then turned on the TDLeaf(λ) learning algorithm, with $\lambda = 0.7$ and the learning rate $\alpha = 1.0$. The value of λ was chosen heuristically, based on the typical delay in moves before an error takes effect, while α was set high enough to ensure rapid modification of the parameters. A couple of minor modifications to the algorithm were made:

- The raw (linear) leaf node evaluations $\tilde{J}(x_i^l, w)$ were converted to a score between -1 and 1 by computing

$$v_i^l := \tanh \left[\beta \tilde{J}(x_i^l, w) \right].$$

This ensured small fluctuations in the relative values of leaf nodes did not produce large temporal differences (the values v_i^l were used in place of $\tilde{J}(x_i^l, w)$ in the TDLeaf(λ) calculations). The outcome of the game $r(x_N)$ was set to 1 for a win, -1 for a loss and 0 for a draw. β was set to ensure that a value of $\tanh \left[\beta \tilde{J}(x_i^l, w) \right] = 0.25$ was equivalent to a material superiority of 1 pawn (initially).

- The temporal differences, $d_t = v_{t+1}^l - v_t^l$, were modified in the following way. Negative values of d_t were left unchanged as any decrease in the evaluation from one position to the next can be viewed as mistake. However, positive values of d_t can occur simply because the opponent has made a blunder. To avoid KnightCap trying to learn to predict its opponent's blunders, we set all positive temporal differences to zero unless KnightCap predicted the opponent's move⁴

³the standard deviation for all ratings reported in this section is about 50

⁴In a later experiment we only set positive temporal differences to zero if KnightCap did not predict the opponent's move and the opponent was rated less than KnightCap. After all, predicting a stronger opponent's blunders is a useful skill, although whether this made any difference is not clear.

- The value of a pawn was kept fixed at its initial value so as to allow easy interpretation of weight values as multiples of the pawn value (we actually experimented with not fixing the pawn value and found it made little difference: after 1764 games with an adjustable pawn its value had fallen by less than 7 percent).

Within 300 games KnightCap's rating had risen to 2150, an increase of 500 points in three days, and to a level comparable with human masters. At this point KnightCap's performance began to plateau, primarily because it does not have an opening book and so will repeatedly play into weak lines. We have since implemented an opening book learning algorithm and with this KnightCap now plays at a rating of 2400–2500 (peak 2575) on the other major internet chess server: ICC, chessclub.com⁵ It often beats International Masters at blitz. Also, because KnightCap automatically learns its parameters we have been able to add a large number of new features to its evaluation function: KnightCap currently operates with 5872 features (1468 features in four stages: opening, middle, ending and mating⁶. With this extra evaluation power KnightCap easily beats versions of Crafty restricted to search only as deep as itself. However, a big caveat to all this optimistic assessment is that KnightCap routinely gets crushed by faster programs searching more deeply. It is quite unlikely this can be easily fixed simply by modifying the evaluation function, since for this to work one has to be able to predict tactics statically, something that seems very difficult to do. If one could find an effective algorithm for "learning to search selectively" there would be potential for far greater improvement.

Note that we have twice repeated the learning experiment and found a similar rate of improvement and final performance level. The rating as a function of the number of a games from one of these repeat runs is shown in figure 4 (we did not record this information in the first experiment). Note that in this case KnightCap took nearly twice as long to reach the 2150 mark, but this was partly because it was operating with limited memory (8Mb) until game 500 at which point the memory was increased to 40Mb (KnightCap's search algorithm—MTD(f) [3]—is a memory intensive variant of α - β and when learning KnightCap must

⁵There appears to be a systematic difference of around 200–250 points between the two servers, so a peak rating of 2575 on ICC roughly corresponds to a peak of 2350 on FICS. We transferred KnightCap to ICC because there are more strong players playing there.

⁶In reality there are not 1468 independent "concepts" per stage in KnightCap's evaluation function as many of the features come in groups of 64, one for each square on the board (like the value of placing a rook on a particular square, for example)

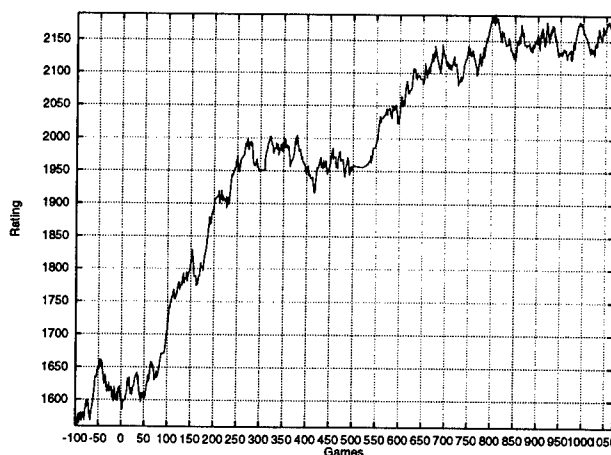


Figure 4: KnightCap's rating as a function of games played (second experiment). Learning was turned on at game 0.

store the whole position in the hash table so small memory really hurts the performance). Another reason may also have been that for a portion of the run we were performing parameter updates after every four games rather than every game.

Plots of various parameters as a function of the number of games played are shown in Figure 5 (these plots are from the same experiment in figure 4). Each plot contains three graphs corresponding to the three different stages of the evaluation function: opening, middle and ending⁷.

Finally, we compared the performance of KnightCap with its learnt weight to KnightCap's performance with a set of hand-coded weights, again by playing the two versions on ICC. The hand-coded weights were close in performance to the learnt weights (perhaps 50–100 rating points worse). We also tested the result of allowing KnightCap to learn starting from the hand-coded weights, and in this case it seems that KnightCap performs better than when starting from just material values (peak performance was 2632 compared to 2575, but these figures are very noisy). We are conducting more tests to verify these results. However, it should not be too surprising that learning from a good quality set of hand-crafted parameters is better than just learning from material parameters. In particular, some of the handcrafted parameters have very high values (the value of an "unstoppable pawn", for example) which can take a very long time to learn under normal playing conditions, particularly if they are rarely active in the principal leaves. It is

⁷KnightCap actually has a fourth and final stage "mating" which kicks in when all the pieces are off, but this stage only uses a few of the coefficients (opponent's king mobility and proximity of our king to the opponent's king).

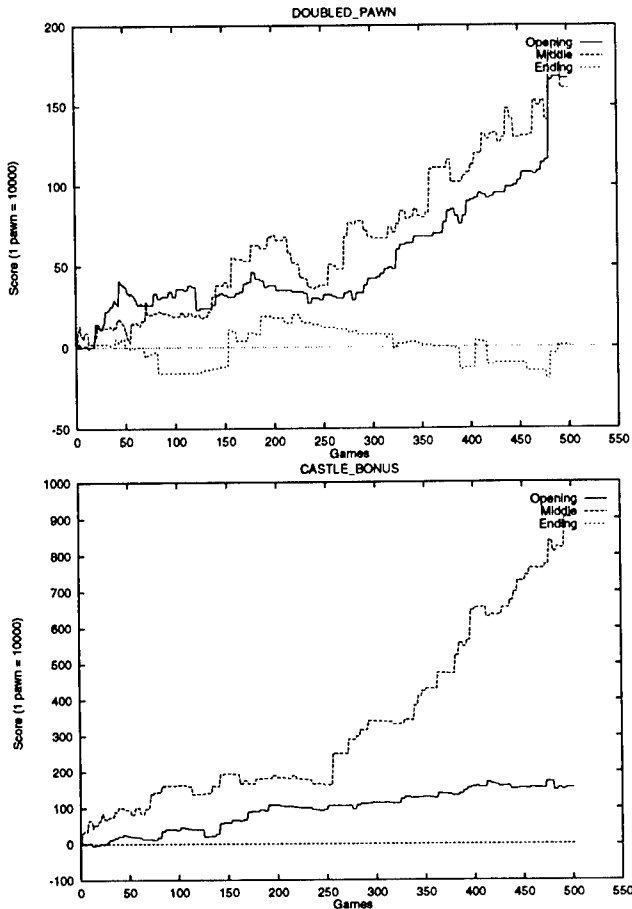


Figure 5: Evolution of two parameters (bonus for castling and penalty for a doubled pawn) as a function of the number of games played. Note that each parameter appears three times: once for each of the three stages in the evaluation function.

not yet clear whether given a sufficient number of games this dependence on the initial conditions can be made to vanish.

4.2 Discussion

There appear to be a number of reasons for the remarkable rate at which KnightCap improved.

1. As all the non-material weights were initially zero, even small changes in these weights could cause very large changes in the relative ordering of materially equal positions. Hence even after a few games KnightCap was playing a substantially better game of chess.
2. It seems to be important that KnightCap started out life with intelligent material parameters. This put it

close in parameter space to many far superior parameter settings.

3. Most players on FICS prefer to play opponents of similar strength, and so KnightCap's opponents improved as it did. This may have had the effect of *guiding* KnightCap along a path in weight space that led to a strong set of weights.
4. KnightCap was learning on-line, not by self-play. The advantage of on-line play is that there is a great deal of information provided by the opponent's moves. In particular, against a stronger opponent KnightCap was being shown positions that 1) could be forced (against KnightCap's weak play) and 2) were mis-evaluated by its evaluation function. Of course, in self-play KnightCap can also discover positions which are mis-evaluated, but it will not find the kinds of positions that are relevant to strong play against other opponents. In this setting, one can view the information provided by the opponent's moves as partially solving the "exploration" part of the *exploration/exploitation* tradeoff.

To further investigate the importance of some of these reasons, we conducted several more experiments.

Good initial conditions.

A second experiment was run in which KnightCap's coefficients were all initialised to the value of a pawn. The value of a pawn needs to be positive in KnightCap because it is used in many other places in the code: for example we deem the MTD search to have converged if $\alpha < \beta + 0.07 \cdot \text{PAWN}$. Thus, to set all parameters equal to the same value, that value had to be a pawn.

Playing with the initial weight settings KnightCap had a blitz rating of around 1250. After more than 1000 games on FICS KnightCap's rating has improved to about 1550, a 300 point gain. This is a much slower improvement than the original experiment. We do not know whether the coefficients would have eventually converged to good values, but it is clear from this experiment that starting near to a good set of weights is important for fast convergence. An interesting avenue for further exploration here is the effect of λ on the learning rate. Because the initial evaluation function is completely wrong, there would be some justification in setting $\lambda = 1$ early on so that KnightCap only tries to predict the outcome of the game and not the evaluations of later moves (which are extremely unreliable).

Self-Play

Learning by self-play was extremely effective for TD-

Gammon, but a significant reason for this is the randomness of backgammon which ensures that with high probability different games have substantially different sequences of moves, and also the speed of play of TD-Gammon which ensured that learning could take place over several hundred-thousand games. Unfortunately, chess programs are slow, and chess is a deterministic game, so self-play by a deterministic algorithm tends to result in a large number of substantially similar games. This is not a problem if the games seen in self-play are "representative" of the games played in practice, however KnightCap's self-play games with only non-zero material weights are very different to the kind of games humans of the same level would play.

To demonstrate that learning by self-play for KnightCap is not as effective as learning against real opponents, we ran another experiment in which all but the material parameters were initialised to zero again, but this time KnightCap learnt by playing against itself. After 600 games (twice as many as in the original FICS experiment), we played the resulting version against the good version that learnt on FICS for a further 100 games with the weight values fixed. The self-play version scored only 11% against the good FICS version.

Simultaneously with the work presented here, Beal and Smith [1] reported positive results using essentially TDLeaf(λ) and self-play (with some random move choice) when learning the parameters of an evaluation function that only computed material balance. However, they were not comparing performance against on-line players, but were primarily investigating whether the weights would converge to "sensible" values at least as good as the naive (1, 3, 3, 5, 9) values for (pawn, knight, bishop, rook, queen) (they did, within 2000 games, and using a value of $\lambda = 0.95$ which supports the discussion in "good initial conditions" above).

5 Conclusion

We have introduced TDLeaf(λ), a variant of TD(λ) suitable for training an evaluation function used in minimax search. The only extra requirement of the algorithm is that the leaf-nodes of the principal variations be stored throughout the game.

We presented some experiments in which a chess evaluation function was trained from B-grade to master level using TDLeaf(λ) by on-line play against a mixture of human and computer opponents. The experiments show both the importance of "on-line" sampling (as opposed to self-play) for a deterministic game such as chess, and the need to start near a good solution for fast convergence, although just how near is still not clear.

On the theoretical side, it has recently been shown that TD(λ) converges for linear evaluation functions[11] (although only in the sense of prediction, not control). An interesting avenue for further investigation would be to determine whether TDLeaf(λ) has similar convergence properties.

Acknowledgements

Thanks to several of the anonymous referees for their helpful remarks. Jonathan Baxter was supported by an Australian Postdoctoral Fellowship. Lex Weaver was supported by an Australian Postgraduate Research Award.

References

- [1] D. F. Beal and M. C. Smith. Learning Piece values Using Temporal Differences. *Journal of The International Computer Chess Association*, September 1997.
- [2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [3] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Best-First Fixed-Depth Minmax Algorithms. *Artificial Intelligence*, 87:255–293, 1996.
- [4] J. Pollack, A. Blair, and M. Land. Coevolution of a Backgammon Player. In *Proceedings of the Fifth Artificial Life Conference*, Nara, Japan, 1996.
- [5] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.
- [6] N. Schraudolph, P. Dayan, and T. Sejnowski. Temporal Difference Learning of Position Evaluation in the Game of Go. In J. Cowan, G. Tesauro, and J. Alspec-tor, editors, *Advances in Neural Information Processing Systems 6*, San Francisco, 1994. Morgan Kaufmann.
- [7] R. Sutton. Learning to Predict by the Method of Temporal Differences. *Machine Learning*, 3:9–44, 1988.
- [8] G. Tesauro. Practical Issues in Temporal Difference Learning. *Machine Learning*, 8:257–278, 1992.
- [9] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.
- [10] S. Thrun. Learning to Play the Game of Chess. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, San Francisco, 1995. Morgan Kaufmann.

- [11] J. N. Tsitsikilis and B. V. Roy. An Analysis of Temporal Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.
- [12] S. Walker, R. Lister, and T. Downs. On Self-Learning Patterns in the Othello Board Game by the Method of Temporal Differences. In C. Rowles, H. Iiu, and N. Foo, editors, *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence*, pages 328–333, Melbourne, 1993. World Scientific.

Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets

Stephen D. Bay*

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697, USA
sbay@ics.uci.edu

Abstract

Combining multiple classifiers is an effective technique for improving accuracy. There are many general combining algorithms, such as Bagging or Error Correcting Output Coding, that significantly improve classifiers like decision trees, rule learners, or neural networks. Unfortunately, many combining methods do not improve the nearest neighbor classifier. In this paper, we present MFS, a combining algorithm designed to improve the accuracy of the nearest neighbor (NN) classifier. MFS combines multiple NN classifiers each using only a random subset of features. The experimental results are encouraging: On 25 datasets from the UCI Repository, MFS significantly improved upon the NN, k nearest neighbor (kNN), and NN classifiers with forward and backward selection of features. MFS was also robust to corruption by irrelevant features compared to the kNN classifier. Finally, we show that MFS is able to reduce both bias and variance components of error.

1 INTRODUCTION

The nearest neighbor (NN) classifier is one of the oldest and simplest methods for performing general, non-parametric classification. It can be represented by the following rule: to classify an unknown pattern, choose the class of the nearest example in the training set as measured by a distance metric. A common extension

is to choose the most common class in the k nearest neighbors (kNN).

Despite its simplicity, the NN classifier has many advantages over other methods. For example, it can learn from a small set of examples, can incrementally add new information at runtime, and often gives competitive performance with more modern methods such as decision trees or neural networks.

Since its inception by Fix and Hodge (1951), researchers have investigated many methods for improving the NN classifier, but most work has concentrated on changing the distance metric or manipulating the patterns in the training set (Dasarathy, 1991). Recently, researchers have begun experimenting with general algorithms for improving classification accuracy by combining multiple versions of a single classifier, also known as a *multiple model* or *ensemble* approach. The outputs of several classifiers are combined in the hope that the accuracy of the whole is greater than the parts. Unfortunately, many combining methods do not improve the NN classifier at all.

For example, in Breiman's (1996) experiments with Bagging, he found no difference in accuracy between the bagged NN classifier and the single model approach. His results suggest that other combining methods that involve any significant degree of resampling or replication of patterns will not work with the NN classifier. Kong and Dietterich (1996) also concluded that Error Correcting Output Coding (ECOC), a method of combining classifiers by decomposing multi-class problems into multiple two-class problems, will not improve classifiers that use local information because of high error correlation. For example, with the NN classifier we predict the class of the closest pattern. This pattern is the same in all of the two-class problems, and hence if it gives an incorrect prediction, all the predictions in the ECOC ensemble will be in-

*Research performed while at the University of Waterloo, Department of Systems Design Engineering, Waterloo, Ont., N2L 3G1, Canada.

correct¹.

In this paper, we present a new method of combining nearest neighbor classifiers with the goal of improving classification accuracy. Our approach manipulates the features that the individual classifiers use. In contrast, other combining algorithms may manipulate the training patterns (Bagging, Boosting) or the class labels (ECOC).

In the next section, we describe the MFS algorithm for combining multiple NN classifiers. In Section 3, we evaluate the algorithm on datasets from the UCI Repository for accuracy, computational complexity, and robustness to irrelevant features. In Section 4, we analyze the algorithm's bias and variance components of error. In Section 5, we discuss related work, and follow it by conclusions and future work in Section 6.

2 CLASSIFICATION FROM MULTIPLE FEATURE SUBSETS

We start by describing the MFS algorithm and then we discuss the motivation behind it and the dangers in using it. We then explain how we set the algorithm's parameters.

2.1 THE MFS ALGORITHM

The algorithm for nearest neighbor classification from multiple feature subsets (MFS) is simple and can be stated as:

Using simple voting, combine the outputs from multiple NN classifiers, each having access only to a random subset of features.

We select the random subset of features by sampling from the original set. We use two different sampling functions: sampling with replacement, and sampling without replacement. In sampling with replacement, a feature can be selected more than once which is equivalent to increasing its weight.

Each of the NN classifiers uses the same number of features. This is a parameter of the algorithm which we set by cross-validation performance estimates on a tuning dataset (see Section 2.2). Each time a pattern

is presented for classification, we select a new random subset of features for each classifier.

As an example of MFS classification, consider Fisher's iris plant classification problem (Fisher, 1936; Duda and Hart, 1973). In this domain, we try to classify iris plants into their specific species: iris-setosa, iris-virginica, and iris-versicolor, based on the following four features: petal length, petal width, sepal length, and sepal width. With MFS we might use three NN classifiers each using a random subset of features. The first NN classifier might use {petal length, sepal width, sepal length}, the second might use {petal width, petal length, sepal width}, and the third might use {petal width, sepal width, sepal width} which we would treat as {petal width, $2 \times$ sepal width}.

The idea of using only a random subset of features may seem counter intuitive, as we are throwing away potentially valuable information. The accuracy of the NN classifiers is likely to decrease compared to a classifier that has access to all the features. Should we not use all the information and make each classifier as accurate as possible? Why should we create a set of classifiers each less accurate than a single one trained on all the information?

The answer to these questions lies in the dynamics of simple voting among a set of classifiers. The individual models do not need to be very accurate for the system as a whole to achieve high accuracy, if the models make different errors. In particular, Hansen and Salamon (1990) showed that under simple voting if the models make independent errors, then the overall error will decrease monotonically with increasing numbers of classifiers. Ali and Pazzani (1996) verified empirically that combining models with uncorrelated errors could significantly reduce the overall error. Selecting different features is an attempt to force the NN classifiers to make different and uncorrelated errors. We are trading off accuracy for error diversity.

There is no guarantee that using different feature sets for the NN classifiers will decorrelate error. However, Tumer and Ghosh (1996) found that with neural networks, selectively removing features could decorrelate errors. Unfortunately, the error rates in the individual classifiers increased, and as a result there was little or no improvement in the ensemble. Cherkauer (1996) was more successful, and was able to combine neural networks that used different hand selected features to achieve human expert level performance in identifying volcanoes from images.

¹Recently Ricci and Aha (1998) have developed a method for combining NN classifiers and ECOC which solves the correlation problem. We discuss this in section 5.

One method of generating a diverse ensemble of classifiers is to perturb some aspect of the training inputs for which the classifier is unstable. For example, Bagging (Breiman, 1996) perturbs the training patterns available to each classifier in the ensemble. Since decision trees are unstable to the patterns, Bagging generates a diverse and effective ensemble. Nearest neighbor classifiers are stable to the patterns, so Bagging generates poor NN ensembles. Nearest Neighbor classifiers, however, are extremely sensitive to the features used. For example, Langley and Iba (1993) found that adding just a few irrelevant features could drastically change the NN classifier's outputs (and reduce accuracy). MFS attempts to use this instability to generate a diverse set of NN classifiers with uncorrelated errors.

The above discussion hopefully provides motivation for why we expect that MFS will improve the accuracy of the nearest neighbor classifier. However, there are three major dangers that we should be aware of when using MFS:

1. Simple voting can only improve accuracy if the classifiers select the correct class more often than any other class. Breiman refers to this as *order correctness*. If the classifiers are not order correct, then simple voting will increase the expected error. For two class problems, we require slightly more than 50% accuracy in the voting classifiers to improve accuracy. With multiple classes, the required accuracy may drop as low as $\frac{1}{C}$ where C is the number of classes.
2. The Bayes error rate can only increase by using a subset of features. This may make it difficult for the NN classifiers used by MFS to meet the requirements in point 1. For example, in the parity problem, a domain with highly interacting features, the Bayes error rate in any proper subset of features is 50% (as opposed to 0% for the full feature space). There is no guarantee that random subsets will have the necessary information for accurate classification.
3. By using the nearest neighbor classifier in the MFS scheme we lose its asymptotic optimality properties. Specifically, as the number of training examples approaches infinity the NN classifier is bounded by twice the Bayes error rate (Cover, 1967). The kNN classifier is Bayes optimal in the limit with proper choice of k (Fix and Hodges, 1951). We can make no such claims about MFS.

2.2 PARAMETER SELECTION

The MFS algorithm has two parameter values that need to be set: the size of the feature subsets, and the number of classifiers to combine.

We set MFS's subset size parameter based on cross-validation accuracy estimates on the training set for the entire ensemble. We evaluated ten evenly spaced intervals over the size of the original feature set. For example, if a domain had 34 features then the subset sizes at 3, 7, 10, ..., 34 were evaluated. In the case of ties, the smaller value was chosen.

We set the number of classifiers by evaluating the performance of MFS on seven development datasets varying the number of classifiers from 10 to 1000. Based on the results, we set the number of classifiers to 100 as a reasonable trade-off between computational expense and accuracy.

3 EXPERIMENTS

3.1 METHODS

We evaluated the performance of MFS using two different sampling functions: sampling with replacement (MFS1) and sampling without replacement (MFS2). We compared these to four other algorithms: nearest neighbor (NN), k nearest neighbor (kNN), nearest neighbor with forward (FSS) and backward (BSS) sequential selection of features (Aha and Bankert, 1994).

The use of FSS and BSS should provide an interesting contrast with MFS. FSS and BSS try to find a single good subset of features, while MFS uses multiple random subsets without regard to their performance.

All classifiers used unweighted Euclidean distance for continuous features and Hamming distance for symbolic features. Missing values were treated as informative and considered to be a specific symbolic value. In the case of continuous features (normalized to [0,1]), a missing value is considered to have a distance of 1 to all non missing values. For the kNN classifier, the value of k was set using cross-validation performance estimates on the training set. For feature selection, we used cross-validation accuracy on the training set for our objective function (also known as a wrapper approach (Kohavi and John, 1996)).

We evaluated the algorithms on twenty-five datasets from the UCI Repository of Machine Learning Databases (Merz and Murphy, 1998). We first normalized the datasets so that continuous features ranged

from $[0, 1]$, and then we ran thirty trials where the training set contained $2/3$ of the patterns (randomly selected) and the test set contained the remaining $1/3$.

There were a few exceptions to this procedure. For Waveform, we used 300 training cases and 4700 test cases to maintain consistency with reported results (Quinlan, 1996). For Satimage, we used the original division into a training and test set, so the results represent one run of each algorithm. For the Musk dataset, which has 166 features, FSS and BSS took too long to run (over 24 hours for a single trial) and no results were obtained.

3.2 ACCURACY

The accuracy and parameter selection results (average k or number of features selected) are shown in Table 1. The first seven datasets were used in the development of the MFS algorithm. The default accuracy is the frequency of the most common class.

The results show that MFS is promising: MFS1 and MFS2 were about 2% more accurate over all domains than it's nearest competitor kNN. MFS1 was best on 16 domains out of 25 (not including MFS2). MFS2 was best on 14 domains and tied in 3 (not including MFS1). For a formal comparison, we used the Wilcoxon signed rank test and found that MFS1 and MFS2 were significantly better than all others with a confidence level greater than 99%.

MFS only performed poorly on two datasets: Iris and Tic-Tac-Toe. For Iris, both MFS1 and MFS2 gave the lowest accuracy out of all the classifiers. This can possibly be explained by the small number of features in the Iris dataset. With only four features, many of the feature subsets would be identical. This would lead to identical errors and high error correlation. For Tic-Tac-Toe, MFS1 performed extremely poorly, having an error rate almost five times that of the NN and kNN classifiers. MFS1 probably performed poorly because in the Tic-Tac-Toe domain the features have a high amount of interaction. We need to examine all the features to determine which side has won. Taking a random subset of features does not make sense and would probably lead to a greatly increased Bayes error rate for the individual classifiers. MFS2 did not experience the same degradation as MFS1 because sampling without replacement degenerated into selecting all the features and hence performing identically to NN.

Comparing MFS1 to MFS2, it is not clear which classifier performed better. MFS1 was better than MFS2 on 15 domains, worse on 7, and tied in 3. However,

MFS2 had a slightly better average accuracy as it did not have a catastrophic failure on Tic-Tac-Toe. The Wilcoxon test did not detect a significant difference between them.

3.3 COMPUTATIONAL COMPLEXITY

The nearest neighbor classifier is often criticized for slow runtime performance, so we will briefly comment on the complexity of MFS and then present actual running times from the experiments.

The NN classifier computes the distance between the test pattern and every pattern in the training set. This requires $O(ef)$ time, where e is the number of examples, and f is the number of features. For MFS, we use n NN classifiers, so its complexity is $O(nef)$. For training, we use cross-validation and MFS requires $O(ne^2fv)$ time, where v is the number of folds (Bay, 1997).

This analysis shows how the computational requirements of MFS change as a function of the number of examples and features. However, it does not give any indication of actual running times on real datasets. Therefore in Table 2 we list the actual running times on an Intel Pentium Pro processor for NN and MFS on the three slowest datasets.

Table 2: Time Requirements for NN and MFS1

Domain	Classification		Training
	NN	MFS1	MFS1
Satimage	0.080s/pat	0.415s/pat	4.6h
Segment	0.015s/pat	0.075s/pat	19.9m
Annealing	0.018s/pat	0.073s/pat	5.5m

Note that even though we are combining 100 classifiers in MFS, it was only about five times as slow as the NN classifier. We attribute this speed up to caching the difference in feature values between the test pattern and all patterns in the training set (i.e. in $d(\mathbf{x}, \mathbf{y}) = (\sum_f (x_f - y_f)^2)^{1/2}$, we cache $(x_f - y_f)^2$).

3.4 ROBUSTNESS TO IRRELEVANT FEATURES

A major drawback of the NN classifier is its sensitivity to irrelevant features. This concerns us because the MFS algorithm uses multiple NN classifiers and hence raises the question: how will the ensemble behave? If the accuracy of the individual NN classifiers drops too low, simple voting can increase the error rate. Since

Table 1: Accuracy and Parameter Selection Results (average k or number of features selected)

Domain	Pat/F	Def.	NN	Accuracy					Average Parameter Settings				
				kNN	FSS	BSS	MFS1	MFS2	kNN	FSS	BSS	MFS1	MFS2
Glass	214/9	35.5	67.9	66.8	72.3	72.5	75.8	76.1	1.7	4.8	5.5	4.4	3.6
Hepatitis	155/19	79.4	79.2	80.4	80.3	77.2	82.7	82.6	6.7	2.4	12.8	8.1	7.0
Ionosphere	351/34	64.1	86.5	85.5	88.2	87.9	93.5	92.7	1.8	4.6	21.9	6.9	6.5
Iris	150/4	33.3	94.3	95.1	93.7	93.5	92.5	92.7	6.1	1.4	2.3	2.8	2.8
Liver-Disorders	345/7	58.0	60.4	61.3	56.8	60.0	65.4	64.4	9.7	1.9	4.2	4.1	3.2
Pima Diabetes	768/8	65.1	69.7	73.6	67.7	68.5	72.5	72.3	11.5	2.0	6.5	4.8	4.2
Sonar	208/60	53.4	85.0	85.1	76.0	84.3	87.3	87.0	1.1	6.3	38.2	15.4	13.2
Annealing	898/38	76.2	98.0	98.0	98.8	98.8	98.6	98.6	1.0	8.2	9.0	31.6	21.3
Automobile	205/25	32.7	70.9	70.9	74.2	72.8	72.5	73.3	1.0	3.3	10.3	8.7	6.3
Breast Cancer	286/9	70.3	65.9	74.3	71.0	70.0	74.0	74.0	8.0	1.9	5.0	6.7	4.6
Credit	690/15	55.5	81.6	85.5	85.7	81.6	86.3	85.8	12.4	3.2	10.5	8.8	6.3
German	1000/20	70.0	70.5	73.1	70.6	68.8	74.4	74.2	10.8	3.0	15.7	15.4	11.2
Horse Colic	368/22	63.0	76.8	79.8	83.9	76.5	80.2	79.8	15.1	2.4	14.8	9.8	7.8
Labor	57/16	64.9	92.1	90.4	78.6	89.5	94.2	94.6	2.3	2.8	7.5	6.7	5.1
Lymphography	148/18	54.7	74.6	77.0	74.8	76.7	81.9	80.4	8.7	3.7	12.1	11.6	8.3
Musk	476/166	56.5	84.3	83.9	na	na	88.9	88.6	1.4	na	na	18.1	19.1
Primary-Tumor	339/17	24.5	37.0	43.5	37.8	38.9	44.5	45.0	13.8	6.3	11.2	10.6	8.1
Satimage	6435/36	22.8	89.5	90.4	88.0	89.4	91.5	91.0	3	10	33	14	11
Segment	2310/19	14.3	93.5	93.0	96.5	96.6	96.8	96.6	4.6	4.8	9.9	10.3	7.9
Soybean-Large	683/35	13.0	90.7	90.5	93.2	90.7	93.4	93.2	1.5	11.9	20.2	21.9	14.9
Tic-Tac-Toe	958/9	65.3	98.1	98.1	87.8	98.1	91.1	98.1	1.0	6.6	9.0	9.0	9.0
Vehicle	946/18	25.8	68.1	67.7	66.6	70.4	71.4	71.4	5.7	5.4	12.5	9.7	6.8
Vote	435/16	54.8	92.9	93.1	95.8	94.6	94.9	94.5	4.3	2.8	9.2	11.8	8.4
Waveform	5000/21	33.9	74.9	81.4	70.3	74.4	81.0	80.9	13.7	7.4	16.8	10.0	8.1
Wine	178/13	39.9	95.2	96.7	92.8	94.8	97.6	97.9	9.8	4.1	7.8	3.8	3.5
average		49.1	79.9	81.4	79.2	80.3	83.3	83.4	6.3	4.6	12.8	10.6	8.3

we are unsure of how the ensemble will behave, we experimentally investigated the robustness of MFS to irrelevant features.

We used the same basic procedure in Section 3.1. We added 10, 20, and 30 boolean irrelevant features to each of the datasets and then measured the accuracy of kNN and MFS1. We chose boolean irrelevant features because they are more difficult for nearest neighbor methods to handle than continuous irrelevant features. This is because while they both have the same range and mean, boolean variables have greater variance.

Table 3 shows the results for several domains. The remaining results (Bay, 1997) are not shown here for space reasons, but they follow a similar pattern.

As expected, irrelevant features always hurt both kNN and MFS to some degree. However, the results are surprising because they reveal that on some domains kNN is critically sensitive while MFS is stable. For example, on Vehicle and Wine with 10 added irrelevant features, kNN drops in accuracy by over 20% while MFS drops by less than 2%. In general, MFS had only minor degradations in accuracy and was occasionally very robust. For example, MFS's accuracy on Iono-

sphere degrades by so little (from 93.5% to 90.1%), it is still better on the dataset corrupted by 30 irrelevant features, than all of the other classifiers on the original dataset.

One possible explanation for MFS's performance lies in how random voters affect the margins of victory in simple voting. For simplicity, let us divide all voters into two types: informed (using relevant features) and uninformed (random) voters. The informed voters cast their ballots, and the winner will have a given margin of votes compared to the next closest competitor. The uninformed, random voters then cast their ballots. The random voters vote with equal probability and equal expectation for all competitors (according to a multinomial distribution). In order for random voting to change the outcome, the number of random votes for class X must meet the following inequality: $randvotes(X) - randvotes(trueclass) > margin(trueclass, X)$. Unless the margins from the informed voters are small, this is unlikely to occur since the $E(randvotes(X)) = E(randvotes(trueclass))$.

As a numerical example, consider a two class problem with fifty informed voters and fifty random voters. The fifty informed voters cast their ballots and the outcome

is 30 votes for class *A* and 20 votes for class *B*. The fifty uninformed voters then cast their ballots. In order for the uninformed voters to change the outcome of the vote (class *A* wins) at least 30 must vote for class *B*. The probability that the decision will change is approximately 8%.

This situation is analogous to what occurs when MFS is applied to domains with irrelevant features. The NN classifiers are the voters, and can become uninformed and random when both of the following conditions are met: (1) the randomly selected features are irrelevant, and (2) the occurrence of the classes in the training set are roughly equal (this is true in many of the UCI datasets). Note that if only the first condition is met, the NN classifier will be random but will choose classes roughly in proportion to their frequencies in the training set.

4 BIAS-VARIANCE ANALYSIS OF ERROR

The expected error of an algorithm can be divided into two components: *bias* which is the consistent error that the algorithm makes over many different runs, and *variance* which is error that fluctuates from run to run. This decomposition is a useful method for explaining how changes to an algorithm affect the final error rates. It allows us to decompose the error into meaningful components and to see how the error components change with variations in the algorithm.

Several researchers have used the bias-variance analysis of error to show how multiple model approaches work. For example, both Breiman (1996b) and Schapire et al. (1997) showed that Bagging improves performance by reducing the variance component of error. Kong and Dietterich (1996) showed that ECOC could reduce both bias and variance.

The bias variance decomposition of error originated in squared error for regression. For classification, 0-1 loss (misclassification rate) is commonly used, but this does not have a straightforward or unique decomposition. Recently, many authors have proposed similar decompositions (Kong and Dietterich, 1996; Breiman, 1996b; James and Hastie, 1997; Tibshirani, 1996; Kohavi and Wolpert, 1996).

We used Kong and Dietterich's (1996) definitions. They define bias to be "the error of the ideal voted hypothesis," which is the result we would get from combining an infinite number of classifiers, each trained on an independent set of examples. Variance is the

"difference between the expected error rate and the ideal voted hypothesis error rate." Formally, where *A* is the algorithm, *m* is the training set size, *x* is the unknown test point, *f*(*x*) is the class of *x*, *f*^{*}(*x*) is the ideal voted hypothesis of the algorithm *A* at *x*, and *Error*(*A*, *m*, *x*) is the expected error of algorithm *A* at *x* using training sets of size *m*, then bias and variance are:

$$Bias(A, m, x) = \begin{cases} 0 & \text{if } f^*(x) = f(x) \\ 1 & \text{if } f^*(x) \neq f(x) \end{cases} \quad (1)$$

$$Variance(A, m, x) = Error(A, m, x) - Bias(A, m, x) \quad (2)$$

Note that the Bayes error is incorporated into the bias error. Also, the variance can be negative. This occurs when the algorithm is usually wrong, but makes a lucky guess and predicts the correct class.

We investigated the bias-variance components of error on three datasets originally used by Breiman (1996b) and later by Schapire et. al (1997) to evaluate multiple model approaches. The datasets are two class problems, with the individual classes composed of 20-dimensional gaussians.

We compared four classifiers: NN, kNN, MFS1 with 1 classifier (1-MFS1), and MFS1 with 100 classifiers. The NN classifier is the control, to which we can compare the kNN and MFS algorithms. 1-MFS1 should allow us to determine the changes to the error components that are caused by random feature selection and the changes that are caused by voting among multiple classifiers.

We used a test set of 3000 instances and 100 independent training sets of size 300 to estimate the bias, variance, and error of the four classifiers. We approximated *f*^{*}(*x*) by voting over the classifiers trained on the 100 independent training sets. The results are shown in Table 4.

In Twonorm and Threenorm, selecting a single random subset of features (1-MFS1) destabilizes the NN classifier and causes the variance error to significantly increase. During voting (MFS1) the variance error is reduced to a much smaller value than the variance of the original NN classifier, thus reducing the overall error significantly.

For Ringnorm, the feature selection process does a dramatic trade of bias for variance. The bias error drops from 47.1% to only 4.6%, while the variance increases

Table 3: Accuracy of kNN and MFS Under Corruption by Irrelevant Features

Domain	kNN				MFS1			
	0	10	20	30	0	10	20	30
Breast Cancer	74.3	71.0	70.3	69.8	74.0	71.5	71.3	70.5
German	73.1	72.0	70.9	70.5	74.4	72.6	71.3	70.7
Ionosphere	85.5	73.7	71.7	69.5	93.5	91.3	91.4	90.1
Soybean-Large	90.5	80.6	75.2	71.1	93.4	87.7	81.2	76.9
Vehicle	67.7	37.8	35.5	34.1	71.4	69.7	66.0	64.2
Vote	93.1	91.8	91.1	90.9	94.9	93.0	92.0	91.3
Wine	96.7	72.5	62.2	61.2	97.6	96.9	93.7	91.8

Table 4: Bias Variance Decomposition of Error

Domain	Opt.	NN	1-MFS1	MFS1	kNN
Twonorm					
bias	2.3	2.4	2.6	2.4	2.4
variance	-	4.9	17.8	1.3	1.0
error	2.3	7.3	20.4	3.7	3.4
Threenorm					
bias	10.5	10.5	11.6	10.4 ²	11.2
variance	-	13.6	22.5	6.3	4.4
error	10.5	24.1	34.1	16.8	15.6
Ringnorm					
bias	1.3	47.1	4.6	3.7	47.1
variance	-	-7.9	25.8	2.0	-7.9
error	1.3	39.2	30.4	5.7	39.2

from -7.9% to 25.8%. Voting then drops the variance to only 2% greatly improving accuracy.

From these datasets, we see that MFS has two modes of operation: (1) decreasing variance through voting, and (2) trading bias for variance through random feature selection. Taken together, MFS is able to reduce both bias and variance components of error.

In comparison to MFS, the kNN classifier reduced only variance. On Twonorm and Threenorm the error of NN was dominated by variance (the bias error was nearly optimal) and like MFS, kNN was able decrease error by reducing the variance. In fact, kNN did a better job than MFS at variance reduction. On Ringnorm, the error of the NN classifier was dominated by bias and kNN was not able to improve performance.

²The value for bias should always be greater than or equal to the Bayes error rate (10.5%), however, because of estimation error from finite sample sizes, it is possible to obtain bias estimates which are lower than the optimal bound.

5 RELATED WORK

Although there is a large body of research on multiple model methods for classification, very little specifically deals with combining NN classifiers. We are only aware of Skalak's (1996) work on combining NN classifiers with small prototype sets, Alpaydin's (1997) work with condensed nearest neighbor (CNN) classifiers (Hart, 1968), and Ricci and Aha's (1998) work on combining NN, feature selection, and ECOC.

Skalak and Alpaydin approach the problem of combining NN classifiers similarly. They drastically reduce the size of each classifier's prototype set to destabilize the NN classifier. Skalak investigates several different strategies for finding a reduced prototype set and even pursues an approach called "radical destabilization" where the NN classifier has just a single prototype per class. He was able to improve accuracy over the baseline NN classifier in 10 of 13 UCI domains. Interestingly, MFS did well on Glass and Lymphography (average increase of over 7% compared to the NN classifier); these are two domains where Skalak reported that no combining algorithm improved performance. Alpaydin uses dataset partitioning (bootstrap or disjoint) in combination with the CNN classifier to edit and reduce the prototypes. He also reported improvements over the NN classifier if the training sets were sufficiently small and thus able to generate diverse classifiers.

Ricci and Aha (1998) applied ECOC to the NN classifier (NN-ECOC). Normally, applying ECOC to NN would not work as the errors in the two-class problems would be highly correlated; however, they found that applying feature selection to the two-class problems decorrelated errors *if different features were selected*. With this method they were able to improve performance in many of the domains tested, and they noted that ECOC accuracy gains tended to increase with in-

creased diversity among the features selected for the two-class problems.

NN-ECOC is similar to MFS as they both use NN classifiers with different features. They differ in that NN-ECOC uses active selection of features (and output coding) while MFS uses random selection. A head to head comparison would be useful to determine if NN-ECOC and MFS achieve their accuracy gains in the same areas of the feature space. Ricci and Aha also analyzed NN-ECOC for bias and variance and concluded that NN-ECOC reduces bias but slightly increases variance. Unfortunately, because we used different a definition of bias and variance our results are not directly comparable.

Regardless of which method has better accuracy, MFS appears to have two main advantages over NN-ECOC: (1) MFS is the simpler algorithm, and (2) MFS is not constrained by ECOC to multiclass problems.

6 CONCLUSIONS AND FUTURE WORK

We introduced MFS, a new algorithm for combining multiple NN classifiers. In MFS, each NN classifier has access to all the patterns in the original training set but only to a random subset of the features.

Our experiments showed that MFS was effective in improving accuracy. But beyond accuracy improvements, MFS is a significant advance because it allows us to incorporate many desirable properties of the NN classifier in a multiple model framework. For example, one of the primary advantages of the NN classifier is its ability to incrementally add new data (or remove old data) without requiring retraining. MFS maintains this property and new data can be added (old data removed) at runtime. Another useful property of the NN classifier is its ability to predict directly from the training data without using intermediate structures. As a result, no matter how many classifiers we combine in MFS, we require only the same memory as a single NN classifier. (The combined NN classifiers can share a common dataset, and the features are selected randomly at runtime.)

MFS has disadvantages and it should not be used indiscriminantly. In particular, MFS loses the asymptotic optimality properties of the NN and kNN classifiers. Additionally, on domains with highly interacting features, such as Tic-Tac-Toe, the error rate can increase too much in the feature subsets resulting in poor ensemble performance. As with all multiple model ap-

proaches, we lose comprehensibility compared to a single model. The individual must judge if the potential accuracy increases is worth these disadvantages.

MFS is our first attempt at using random feature selection to generate effective NN ensembles, and although successful at improving accuracy, there are still many unanswered questions and open areas for future work:

1. *Why does MFS work?* We made an initial attempt at answering this question with our analysis of irrelevant features and the bias-variance decomposition of error. But clearly more work needs to be done as we cannot even characterize the domains MFS will do well on.
2. *Application to other classifiers.* We showed that random feature selection is useful for generating ensembles of NN classifiers. Can we apply this technique to other learning algorithms?
3. *Implications for feature selection and feature weighting.* The experimental results showed that combining multiple random feature subsets can significantly improve performance over the single best subset of features found by FSS or BSS. This implies that instead of searching for the single best set of features, we should be searching for multiple feature sets that work well together.
4. *Other Improvements.* In this paper, we kept the design of MFS as simple as possible; however, there are a number of obvious improvements that may help accuracy and speed. In particular, we would like to investigate: (1) different weighting schemes, (2) varying the number of features each classifier uses, (3) postpruning the ensemble, (4) combining more sophisticated versions of the NN classifier, and (5) editing the prototypes.

Acknowledgements

I thank Michael Pazzani for his support and encouragement. I also thank Cathy Blake, Yang Wang, and the anonymous reviewers for providing many comments that improved this paper. This work was partially supported by an NSERC PGS A scholarship.

References

- D. W. Aha and R. L. Bankert. (1994). Feature selection for case-based classification of cloud types: An empirical comparison. In *Proceedings of the AAAI-94 Workshop on Case-Based Reasoning*, pages 106–112.

- K. M. Ali and M. J. Pazzani. (1996). Error reduction through learning multiple descriptions. *Machine Learning*, 24:173–202.
- E. Alpaydin. (1997). Voting over multiple condensed nearest neighbors. *Artificial Intelligence Review*, 11(1-5):115–132.
- S. D. Bay. (1997). Nearest neighbour classification from multiple data representations. Master's thesis, University of Waterloo, Department of Systems Design Engineering.
- L. Breiman. (1996). Bagging predictors. *Machine Learning*, 24:123–140.
- L. Breiman. (1996b). Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, Berkeley.
- K. J. Cherkauer. (1996). Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In P. Chan, editor, *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, pages 15–21. Available from <http://www.cs.fit.edu/~imlm>.
- T. M. Cover and P. E. Hart. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.
- B. V. Dasarathy. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA.
- R. O. Duda and P. E. Hart. (1973). *Pattern Classification and Scene Analysis*. John Wiley, New York.
- R. A. Fisher. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188.
- E. Fix and J. L. Hodges. (1951). Discriminatory analysis: Nonparametric discrimination: Consistency properties. Technical Report Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolph Field, Texas.
- L. K. Hansen and P. Salamon. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001.
- P. E. Hart. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516.
- G. James and T. Hastie. (1997). Generalizations of the bias/variance decomposition for prediction error. <http://stat.stanford.edu/~gareth>.
- R. Kohavi and G. H. John. (1996). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.
- R. Kohavi and D. H. Wolpert. (1996). Bias plus variance decomposition for zero-one loss functions. In *Machine Learning: Proceedings of the Thirteenth International Conference*.
- E. B. Kong and T. G. Dietterich. (1996). Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 725–730.
- P. Langley and W. Iba. (1993). Average-case analysis of a nearest neighbor algorithm. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 889–894.
- C. J. Merz and P. M. Murphy. (1998). UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Science. <http://www.ics.uci.edu/~mlearn/>.
- J. R. Quinlan. (1996). Bagging, Boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730.
- F. Ricci and D. W. Aha. (1998). Error-correcting output codes for local learners. In *Proceedings of the 10th European Conference on Machine Learning*.
- R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. In *Machine Learning: Proceedings of the Fourteenth International Conference*.
- D. B. Skalak. (1996). *Prototype Selection for Composite Nearest Neighbor Classifiers*. PhD thesis, Department of Computer Science, University of Massachusetts.
- R. Tibshirani. (1996). Bias, variance and prediction error for classification rules. Technical report, Department of Statistics, University of Toronto.
- K. Tumer and J. Ghosh. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8:385–404. Special issue on combining artificial neural networks: ensemble approaches.

Learning Collaborative Information Filters

Daniel Billsus and Michael J. Pazzani

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
{dbillsus, pazzani}@ics.uci.edu

Abstract

Predicting items a user would like on the basis of other users' ratings for these items has become a well-established strategy adopted by many recommendation services on the Internet. Although this can be seen as a classification problem, algorithms proposed thus far do not draw on results from the machine learning literature. We propose a representation for collaborative filtering tasks that allows the application of virtually any machine learning algorithm. We identify the shortcomings of current collaborative filtering techniques and propose the use of learning algorithms paired with feature extraction techniques that specifically address the limitations of previous approaches. Our best-performing algorithm is based on the singular value decomposition of an initial matrix of user ratings, exploiting latent structure that essentially eliminates the need for users to rate common items in order to become predictors for one another's preferences. We evaluate the proposed algorithm on a large database of user ratings for motion pictures and find that our approach significantly outperforms current collaborative filtering algorithms.

with respect to users' individual preferences. Second, the number of users accessing the Internet is also growing. Not only does this lead to an incredible variety of subjects that can be learned about online, it opens up new possibilities to organize and recommend information. The central idea here is to base personalized recommendations for users on information obtained from other, ideally like-minded, users. This is commonly known as collaborative filtering or social filtering.

The underlying techniques used in today's recommendation systems fall into two distinct categories: content-based and collaborative methods. Content-based methods require textual descriptions of the items to be recommended and draw on results from both information retrieval and machine learning research (e.g., Pazzani and Billsus, 1997). In general, a content-based system analyzes a set of documents rated by an individual user and uses the content of these documents, as well as the provided ratings, to infer a profile that can be used to recommend additional items of interest. In contrast, collaborative methods recommend items based on aggregated user ratings of those items, i.e. these techniques do not depend on the availability of textual descriptions. Both approaches share the common goal of assisting in the user's search for items of interest, and thus attempt to address one of the key research problems of the information age: locating needles in a haystack that is growing exponentially.

1 INTRODUCTION

Research on intelligent information agents in general, and recommendation systems in particular, has recently attracted much attention. The reasons for this are twofold. First, the amount of information available to individuals is growing steadily. Information overload has become a popular buzzword of our times and people feel overwhelmed when navigating through today's information and media landscape. This leads to a clear demand for automated methods, commonly referred to as intelligent information agents, that locate and retrieve information

In this paper we focus on collaborative filtering techniques. A variety of algorithms have previously been reported in the literature and their promising performance has been evaluated empirically (Shardanand and Maes, 1995; Hill et al. 1995; Resnick et al. 1994). These results, and the continuous increase of people connected to the Internet, led to the development and employment of numerous collaborative filtering systems. Virtually all topics that could be of potential interest to users are covered by special-purpose recommendation systems: web pages, news stories, movies, music videos, books, CDs, restaurants, and many more. Some of the best-known represen-

tatives of these systems, such as *FireFly* (www.firefly.com) or *WiseWire* (www.wisewire.com) have turned into commercial enterprises. Furthermore, collaborative filtering techniques are becoming increasingly popular as part of online shopping sites. These sites incorporate recommendation systems that suggest products to users based on products that like-minded users have ordered before, or indicated as interesting. For example, users can find out which CD they should order from an online CD store if they provide information about their favorite artists, and several online bookstores (e.g. amazon.com) can associate their available titles with other titles that were ordered by like-minded people.

Although there seems to be an increasingly strong demand for collaborative filtering techniques, only a few different algorithms have been proposed in the literature thus far. Furthermore, the reported algorithms are based on rather simple predictive techniques. Although collaborative filtering can be seen as a classification task, the problem has not received much attention in the machine learning community. It seems likely that predictive performance can be increased through the development of special-purpose algorithms that draw on results from the machine learning literature.

This paper can be outlined as follows. We briefly present the central ideas of previously reported collaborative filtering algorithms. We identify the main shortcomings of these approaches and motivate the need for techniques that do not suffer from these limitations. We then explain how the task of computing collaborative recommendations can be represented as a classification task. Within this framework we present a learning algorithm that addresses the limitations of previous approaches. The proposed method is based on dimensionality reduction through the singular value decomposition (SVD) of an initial matrix of user ratings, exploiting latent structure that essentially eliminates the need for users to rate common items in order to become predictors for one another's preferences. An artificial neural network is used to compute final recommendations. We evaluate our algorithm on a large database of user ratings for motion pictures and show that it significantly outperforms previously proposed algorithms.

2 COLLABORATIVE FILTERING ALGORITHMS

In this section we briefly outline the main ideas of collaborative filtering algorithms reported in the literature. Shardanand and Maes, 1995, discuss a variety of social filtering algorithms and evaluate them in the context of their music recommendation system *Ringo* (predecessor to *FireFly*). These algorithms are based on a simple intuition: predictions for a user should be based on the simi-

larity between the interest profile of that user and those of other users. Therefore, the first step of these algorithms is to compute similarities between user profiles. Suppose we have a database of user ratings for items, where users indicate their interest in an item on a numeric scale. It is now possible to define similarity measures between two user profiles, U and J , where a user profile simply consists of a vector of numeric ratings. A measure proposed by Shardanand and Maes is the *Pearson correlation coefficient*, r_{UJ} . Once the similarity between profiles has been quantified, it can be used to compute personalized recommendations for users. All users whose similarity is greater than a certain threshold t are identified and predictions for an item are computed as the weighted average of the ratings of those similar users for the item, where the weight is the computed similarity. Note that this prediction scheme leads to cases where predictions cannot be computed for all items in the database. If the threshold t is set to a high value, only a few very similar users are considered and it becomes increasingly likely that ratings for some specific item are not available. In order to avoid this problem, (Resnick et al., 1994) compute predictions according to the following formula, where U_x is a rating to be predicted for User U on item x and r_{UJ} is the correlation between users U and J .

$$U_x = \bar{U} + \frac{\sum_{J \in \text{Raters of } x} (J_x - \bar{J}) r_{UJ}}{\sum_{J \in \text{Raters of } x} |r_{UJ}|}$$

where

$$r_{UJ} = \frac{\sum (U - \bar{U})(J - \bar{J})}{\sqrt{\sum (U - \bar{U})^2 \cdot \sum (J - \bar{J})^2}}$$

If no ratings for item x are available, the prediction is equivalent to the mean of all ratings from user U . Similar algorithms were reported and evaluated in (Hill et al. 1995).

While these correlation-based prediction schemes were shown to perform well, they suffer from several limitations. Here, we identify three specific problems: First, correlation between two user profiles can only be computed based on items that both users have rated, i.e. the summations and averages in the correlation formula are only computed over those items that both users have rated. If users can choose among thousands of items to rate, it is likely that overlap of rated items between two users will be small in many cases. Therefore, many of the computed correlation coefficients are based on just a few observations, and thus the computed correlation cannot be regarded as a reliable measure of similarity. For example, a correlation coefficient based on three observations has as much influence on the final prediction as a coefficient

based on 30 observations. Second, the correlation approach induces one global model of similarities between users, rather than separate models for classes of ratings (e.g. positive rating vs. negative rating). Current approaches measure whether two user profiles are positively correlated, not correlated at all or negatively correlated. However, ratings given by one user can still be good predictors for ratings of another user, even if the two user profiles are not correlated. Consider the case where user A's positive ratings are a perfect predictor for a negative rating from user B. However, user A's negative ratings do not imply a positive rating from user B, i.e. the correlation between the two profiles could be close to zero, and thus potentially useful information is lost. Third, and maybe most importantly, two users can only be similar if there is overlap among the rated items, i.e. if users did not rate any common items, their user profiles cannot be correlated. Due to the enormous number of items available to rate in many domains, this seems to be a serious stumbling block for many filtering services, especially during the startup phase. However, just knowing that users did not rate the same items does not necessarily mean that they are not like-minded. Consider the following example: Users A and B are highly correlated, as are users B and C. This relationship provides information about the similarity between users A and C as well. However, in case users A and C did not rate any common items, a correlation-based similarity measure could not detect any relation between the two users. We believe that potentially useful information is lost if this kind of transitive similarity relation cannot be detected.

3 COLLABORATIVE FILTERING AS A CLASSIFICATION PROBLEM

In this section we present collaborative filtering in a machine learning framework and suggest the use of an algorithm that specifically addresses the aforementioned limitations of correlation-based approaches.

Collaborative filtering can be seen as a classification task. Based on a set of ratings from users for items, we are trying to induce a model for each user that allows us to classify unseen items into two or more classes, for example *like* and *dislike*. Alternatively, if our goal is to predict user ratings on a continuous scale, we have to solve a regression problem.

Our initial data exists in the form of a sparse matrix, where rows correspond to users, columns correspond to items and the matrix entries are ratings. Note that *sparse* in this context means that most elements of the matrix are empty, because every user typically rates only a very small subset of all possible items. The prediction task can now be seen as filling in the missing matrix values. Since we are interested in learning personalized models for each

user, we associate one classifier (or regression model) with every user. This model can be used to predict the missing values for one row in our matrix.

Table 1: Exemplary User Ratings

	I ₁	I ₂	I ₃	I ₄	I ₅
U ₁	4		3		
U ₂		1		2	
U ₃	3	4	2		4
U ₄	4	2	1		?

With respect to Table 1, consider that we would like to predict user 4's rating for item 5. We can train a learning algorithm with the information that we have about user 4's previous ratings. In this example user 4 has provided 3 ratings, which leads to 3 training examples: I_1 , I_2 , and I_3 . These training examples can be directly represented as feature vectors, where users correspond to features (U_1 , U_2 , U_3) and the matrix entries correspond to feature values. User 4's ratings for I_1 , I_2 and I_3 are the class labels of the training examples. However, in this representation we would have to address the problem of many missing feature values. If the learning algorithm to be used cannot handle missing feature values, we can apply a simple transformation. Note that we cannot introduce an additional numeric value that indicates a missing feature, because this would conflate the new value and the observed ratings. However, every user can be represented by up to n Boolean features, where n is the number of points on the scale that is used for ratings. For example, if the full n -point scale of ratings is used to represent ratings from m users, the resulting Boolean features are of the form "User m 's rating was i ", where $0 < i \leq n$. We can now assign Boolean feature values to all of these new features. If this representation leads to an excessive number of features that only appear rarely throughout the data, the rating scale can be further discretized, e.g. into the two classes *like* and *dislike*. The resulting representation is simple and intuitive: a training example E corresponds to an item that the user has rated, the class label C is the user's discretized rating for that item, and items are represented as vectors of Boolean features F_i .

Table 2: Exemplary Feature Vectors

	E ₁	E ₂	E ₃
U ₁ like	1	0	1
U ₁ dislike	0	0	0
U ₂ like	0	0	0
U ₂ dislike	0	1	0
U ₃ like	1	1	0
U ₃ dislike	0	0	1
Class	like	dislike	dislike

Table 2 shows the resulting Boolean feature vectors (*true* = 1 and *false* = 0) for user 4, where a rating of either 1 or 2 corresponds to the class *dislike*, and a rating of either 3 or 4 corresponds to the class *like*.

After converting a data set of user ratings for items into this format, we can draw on the machine learning literature and apply virtually any supervised learning algorithm that, through analysis of a labeled training sample $T = \{E_j, C_j\}$, can induce a function $f: E \rightarrow C$.

However, if we look back at the correlation-based approaches described earlier and express them in our learning framework, we notice that these algorithms solve a classification problem in a somewhat unconventional way. If features and classes are represented as ordinal values (no discretization), these algorithms measure the degree of correlation between features and class labels. Predictions for unseen examples are then computed as a weighted average of feature values. While this approach seems to work reasonably well for the domain at hand, it is not supported by a sound theory that we could use to motivate the algorithms' use for either a classification or regression task. It comes as no surprise that researchers in machine learning have thus far not attempted to solve any task with this algorithm. It seems likely that theoretically well-founded algorithms that have the discrimination between classes as their specific goal, can outperform correlation-based approaches.

3.1 REDUCING DIMENSIONALITY

Our goal is to construct or apply algorithms that address the previously identified limitations of correlation-based approaches. As mentioned earlier, the computation of correlation coefficients can be based on too little information, leading to inaccurate similarity estimates. When applying a learning algorithm, we would like to avoid this problem. In particular, we would like to discard information that we do not consider informative for our classification task. Likewise, we would like to be able to take possible interaction and dependencies among features into account, as we regard this as an essential prerequisite for users to become predictors for one another's preferences even without rating common items. Both of these issues can be addressed through the application of appropriate feature extraction techniques. Furthermore, the need for dimensionality reduction is of particular importance if we represent our data in the proposed learning framework. For large databases containing many users we will end up with thousands of features while our amount of training data is very limited. Learning under these conditions is not practical, because the amount of data points needed to approximate a concept in d dimensions grows exponentially with d , a phenomenon commonly referred to as the *curse of dimensionality* (Bellman, 1961). This is, of course, not a problem unique to collaborative filtering.

Other domains with very similar requirements include the classification of natural language text or, in general, any information retrieval task. In these domains the similarity among text documents needs to be measured. Ideally, two text documents should be similar if they discuss the same subject or contain related information. However, it is often not sufficient to base similarity on the overlap of words. Two documents can very well discuss similar subjects, but use a somewhat different vocabulary. A low number of common words should not imply that the documents are not related. This is very similar to the problem we are facing in collaborative filtering: the fact that two users rated different items should not imply that they are not like-minded. Researchers in information retrieval have proposed different solutions to the text version of this problem. One of these approaches, Latent Semantic Indexing (LSI) (Deerwester et al., 1990) is based on dimensionality reduction of the initial data through singular value decomposition (SVD). We will now show how the SVD can be used as a dimensionality reduction technique for our collaborative filtering task. A more detailed description of underlying algebraic principles can be found in (Berry et al., 1994).

3.2 COLLABORATIVE FILTERING AND THE SVD

We start our analysis based on a rectangular matrix containing Boolean values that indicate user ratings for items (see Table 2). This matrix is typically very sparse, where *sparse* means that most elements are zero, because each item is only rated by a small subset of all users. Furthermore, many features appear infrequently or do not appear at all throughout this matrix. However, features will only affect the SVD if they appear at least twice. Therefore, we apply a first preprocessing step and remove all features that appear less than twice in our training data. The result of this preprocessing step is a matrix A containing zeros and ones, with at least two ones in every row. Using the SVD, the initial matrix A with r rows, c columns and rank m can be decomposed into the product of three matrices:

$$A = U \Sigma V^T$$

where the columns of U and V are orthonormal vectors that define the *left* and *right* singular vectors of A , and Σ is a diagonal matrix containing corresponding singular values. Since the derived vectors are orthonormal, no vector can be reconstructed as a linear combination of the others. U is an $m \times c$ matrix and the singular vectors correspond to columns of the original matrix. V is an $r \times m$ matrix and the singular vectors correspond to rows of the original matrix. The singular values quantify the amount of variance in the original data captured by the singular vectors. This representation provides an ideal framework for dimensionality reduction, because one can now quantify the amount of information that is lost if singular val-

ues and their corresponding singular vector elements are discarded. The smallest singular values are set to zero, reducing the dimensionality of the new data representation. The underlying intuition is that the n largest singular values together with their corresponding singular vector elements capture the important "latent" structure of the initial matrix, whereas random fluctuations are eliminated. The usefulness of the SVD for our task can be further explained by its geometric interpretation. If we choose to retain the k largest singular values, we can interpret the singular vectors, scaled by the singular values, as coordinates of points representing the rows and columns of the original matrix in k dimensions. In our context, the goal of this transformation is to find a spatial configuration such that items and user ratings are represented by points in k -dimensional space, where every item is placed at the centroid of every user rating that it received and every user rating is placed at the centroid of all the items that it was assigned to. While the position of vectors in this k -dimensional space is determined through the assignment of ratings to items, items can still be close in this space even without containing any common ratings. Likewise, user ratings can be close to each other, although they were never assigned to a common set of items. Many different strategies for classification of items are theoretically possible using this k -dimensional representation. We will now describe the complete algorithm for item classification that we used in our experiments.

3.3 USING SINGULAR VECTORS AS TRAINING EXAMPLES

Our training data is a set of rated items, represented as Boolean feature vectors (see Table 2). We compute the SVD of the training data and discard the n smallest singular values, reducing the dimensionality to k . Currently, we set k to $0.9 \cdot m$, where m is the rank of the initial matrix. This value was chosen because it resulted in the best classification performance (evaluated using a tuning set, see Section 4). The singular vectors of matrix U scaled by the remaining singular values represent rated items in k dimensions. These vectors become our new training examples. Since we compute the SVD of the training data, resulting in real-valued feature vectors of size k , we need to specify how we transform examples to be classified into this format. Based on the geometric interpretation of the SVD, the solution to this problem is straightforward. We compute a k -dimensional vector for an item, so that with appropriate rescaling of the axes by the singular values, it is placed at the centroid of all the user ratings that it contains. Mathematically, we can compute this vector as:

$$v_k = v^T U_k \Sigma_k^{-1}$$

where v is a Boolean feature vector containing user ratings, U_k is a matrix of singular vectors with k elements in

each vector, and Σ_k is a diagonal matrix containing the k largest singular values.

At this point we need to pick a suitable learning algorithm that takes real-valued feature vectors as its input and learns a function that either predicts class membership or computes a score a user would assign to an item. Ideally, we would like to use a learning paradigm that allows for maximum flexibility in evaluating this task as either a regression or classification problem. Therefore, we selected artificial neural networks as the method of choice for our purposes (Rumelhart and McClelland, 1986). It can be shown that neural networks with linear output units and a single hidden layer can approximate any continuous function f by increasing the size of the hidden layer (Ripley, 1996). This allows us to solve a regression problem. Alternatively, if we replace the linear output units by logistic units, we can use the same framework to perform logistic regression, or learn to discriminate between classes. We ran various experiments on a tuning set of the data available to us, to determine a network topology and learning paradigm that resulted in good performance (see Section 4 for details on the experimental evaluation). The winning approach was a feed-forward neural network with k input units, 2 hidden units and 1 output unit. The hidden units use sigmoid functions, while the output unit is linear. Weights are learned with backpropagation. Although the task at hand might suggest using a user's rating as the function value to predict, we found that a slightly different approach resulted in better performance. We determined the average rating for an item¹ and trained the network on the difference between a user's rating and the average rating. This function appeared to be easier to learn, presumably because the function values take on extreme values less frequently and in these cases express a user's individual taste. In order to predict scores for items, the output of the network needs to be added to the mean of the item. We then used a threshold t (depending on the rating scale of the domain, see next section) to convert the predicted rating to a binary class label. In summary, our algorithm for collaborative filter induction proceeds in the following steps:

Training:

- Convert the training data, a sparse matrix of user ratings, to Boolean feature vectors, resulting in a matrix filled with zeros (*false*) and ones (*true*).
- Compute the SVD of the training data.
- Select k , the number of dimensions to retain, and reduce the extracted singular vectors accordingly.
- Train a neural network with singular vectors scaled by singular values.

¹ The average is computed using ratings from all users who rated the item, except the user whose rating is to be predicted.

Predicting:

- Convert the item's user ratings to a Boolean feature vector.
- Scale the feature vector into the k-dimensional space.
- Feed the resulting real-valued vector to the trained neural network to compute a prediction.

4 EXPERIMENTAL EVALUATION

In this section we report results of the experimental evaluation of our proposed algorithm. We describe the data set used, the experimental methodology, as well as performance measures we consider appropriate for this task.

4.1 THE EACHMOVIE DATABASE

We ran experiments using data from the *EachMovie* collaborative filtering service. The *EachMovie* service was part of a research project at the Systems Research Center of Digital Equipment Corporation. The service was available for a period of 18 months and was shut down in September 1997. During that time the database grew to a fairly large size, containing ratings from 72,916 users on 1,628 movies. User ratings were recorded on a numeric six-point scale (0.0, 0.2, 0.4, 0.6, 0.8, 1.0). The data set is publicly available and can be obtained from Digital Equipment Corporation (McJones, 1997).

Although data from 72,916 users is available, we restrict our analysis to the first 2,000 users in the database. These 2,000 users provided ratings for 1,410 different movies. We restricted the number of users considered, because we are interested in the performance of the algorithm under conditions where the ratio of users to items is low. This is a situation that every collaborative filtering service has to go through in its startup-phase, and in many domains we cannot expect to have significantly more users than items. We also believe that the deficiencies of correlation-based approaches will be more noticeable under these conditions, because it is less likely to find users with considerable overlap of rated items.

4.2 PERFORMANCE MEASURES

We are most interested in a system that can accurately distinguish between movies a user would like and all other movies rather than a method that accurately predicts the numeric rating of every movie. Of course, a method that predicts the actual ratings most exactly could also be the best classifier for this classification task. To analyze this, we defined two classes, *hot* and *cold*, that were used to label movies. When transforming movies to training examples for a particular user, we label movies as *hot* if

the rating for the movie was 0.8 or 1.0, or *cold* otherwise. We decided to use this threshold since we are interested in identifying movies the user would like and feel strongly about. Since the correlation-based approaches as well as the neural network predict numeric ratings, we base the classification of movies on this numeric prediction, and classify them as *hot* if the predicted rating exceeds the threshold 0.7 (midpoint between the two possible user ratings 0.6 and 0.8). At the same time, we can still use the predicted score to rank-order classified movies. Not only does assigning class labels allow us to measure classification accuracy, we can also apply additional performance measures, such as *precision* and *recall*, commonly used for information retrieval tasks. In our domain, *precision* is the percentage of movies classified as *hot* that are *hot*, and *recall* is the percentage of *hot* movies that were classified as *hot*. We believe that these measures are appropriate for our study, because we would like to quantify performance for a task that has the identification of relevant items as its goal.

It is important to evaluate *precision* and *recall* in conjunction, because it is easy to optimize either one separately. However, for a classifier to be useful for our purposes we demand that it be precise as well as have high recall. In order to quantify this with a single measure, (Lewis and Gale, 1994) proposed the *F-measure*, a weighted combination of precision and recall that produces scores ranging from 0 to 1. Here we assign equal importance to precision and recall:

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In summary, we measure the overall performance of the algorithms using classification accuracy and the F-measure. Since we see the F-measure as a useful construct to compare classifiers, but think that it is not an intuitive measure to indicate a user's perception of the usefulness of an actual system, we use an additional measure: precision at the top n ranked items (here, we report scores for $n = 3$ and $n = 10$).

4.3 EXPERIMENTAL METHODOLOGY

Since we are interested in the performance of the algorithms with respect to the number of ratings provided by users, we report learning curves where we vary the number of rated items from 10 to 50. For each user we ran a total of 30 paired trials for each algorithm. For an individual trial of an experiment, we randomly selected 50 rated items to use as a training set, and 30 as a test set. We then started training with 10 examples out of the set of 50 and increased the training set incrementally in steps of 10, measuring the algorithms' performance on the test set for each training set size. Final results for one user are then averaged over all trials. We repeated this for 20 users and

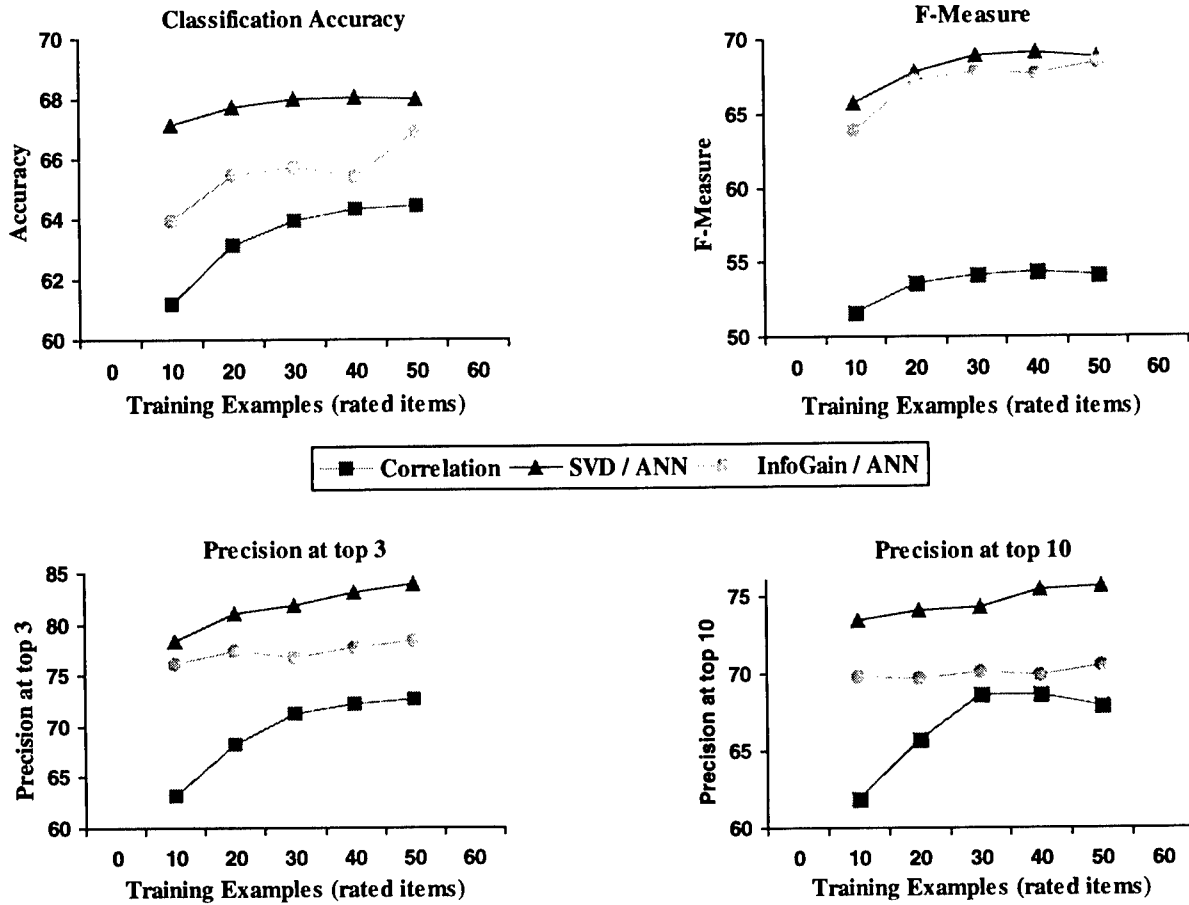


Figure 1: Learning Curves

the final curves reported here are averaged over those 20 users.

The actual size of the feature vectors used to train the neural network depends on the number of rated items in the current training set, as well as the particular rated items. Initially, every training example consists of 4000 Boolean values (2000 users * 2 features per user). Deleting all features that appear less than twice reduces the number of features approximately by a factor of 4 (see section 3.2), i.e. if we start to train our algorithm with 10 examples, we have an initial 10×1000 matrix of training data. After decomposing this matrix using the SVD, the matrix U that represents rated items in a space of lower dimensions is a 10×10 matrix (because the initial matrix has 10 columns and this is also the rank of the matrix). Since we keep only 90% of the singular values, the resulting feature vectors consist of 9 real values. Likewise, if we have 50 examples in the training set, the resulting size of every training example after dimensionality reduction is 45.

We determined parameters for our algorithms using a tuning set of 20 randomly selected users. The results reported here are averaged over 20 different users. The training data for these users is based on ratings from the first 2000 users of the database, as described earlier. We

selected users randomly, but with the following constraints. First, only users whose prior probability of liking a movie is below 0.75 are considered. Otherwise, scores that indicate high precision of our algorithms might be biased by the fact that there are some users in the database who either like everything or just gave ratings for movies they liked. Second, only users that rated at least 80 movies were selected, so that we could use the same number of training and test examples for all users.

4.4 SUMMARY OF RESULTS

Figure 1 summarizes the performance of three different algorithms. The algorithm labeled *Correlation* is the correlation-based approach that performed best on this data out of the strategies described in Section 2. This approach uses the prediction formula as described in (Resnick et al 1994) and summarized in Section 2. We consider all correlations, i.e. we do not require correlations to be above a certain threshold. The algorithm labeled *SVD/ANN* is our dimensionality reduction approach coupled with a neural network as described in Section 3.3. Since this algorithm is a combination of a feature extraction technique (SVD) and a learning algorithm (ANN), the observed performance does not allow us to infer anything about the relative importance of each technique individually. Therefore, we report the performance of a third algorithm, labeled *Info-*

Gain/ANN, in order to quantify the importance of our proposed feature extraction technique. *InfoGain/ANN* uses the same neural network setup as *SVD/ANN*, but applies a different feature selection algorithm. Here, we compute the expected information gain (Quinlan, 1986) of all the initial features and then select the n most informative features, where n is equivalent to the number of features used by *SVD/ANN* for each training set size. Since expected information gain cannot detect interaction and dependencies among features, the difference between *SVD/ANN* and *InfoGain/ANN* allows us to quantify the utility of the SVD for this task.

The results show that both *SVD/ANN*, as well as *InfoGain/ANN*, performed better than the correlation approach. In addition, *SVD/ANN* is more accurate and substantially more precise than *InfoGain/ANN*. At 50 training examples *Correlation* reaches a classification accuracy of 64.4%, vs. 67.9% for *SVD/ANN*. While predictive accuracy below 70% might initially seem disappointing, we need to keep in mind that our goal is not the perfect classification of all movies. We would like to have a system that identifies many interesting items and does this with high precision. This ability is measured by the F-measure and we can see that *SVD/ANN* has a significant advantage over the correlation approach (at 50 examples 54.2% for *Correlation* vs. 68.8% for *SVD/ANN*). Finally, if we restrict our analysis to the top 3 or top 10 suggestions of each algorithm, we can see that *SVD/ANN* is much more precise than the other two algorithms. At 50 training examples *Correlation* reaches a precision of 72.6% at the top 3 suggestions, *InfoGain/ANN*'s precision is 78.3% and *SVD/ANN* reaches 83.9%. These results are encouraging and provide empirical evidence that the use of theoretically well-founded learning algorithms can lead to improved predictive performance on collaborative filtering tasks. Furthermore, we have shown that an additional performance increase can be obtained through the use of appropriate dimensionality reduction techniques, such as the SVD.

5 DISCUSSION AND FUTURE WORK

Our experiments illustrate the potential of dimensionality reduction techniques that exploit the underlying "latent structure" of user ratings. The key to success of this method is that it can utilize information from users whose ratings are not correlated, or who have not even rated anything in common. However, since we are computing the SVD of the training data, i.e. a matrix consisting only of feature vectors for all items a user has rated, we might not be exploiting the full potential of the method. Including feature vectors of items that the user has not rated in the matrix to decompose will affect the position of the singular vectors corresponding to labeled training examples in k -dimensional space. Future experiments will re-

veal if further performance improvements can be achieved through the addition of unlabeled training data.

We believe that additional knowledge about the similarity of users and items can be gained through the analysis of textual descriptions of items. Our long-term goal of this work is to combine collaborative and content-based filtering techniques. Similarity between users could then be influenced by similarity between descriptions of rated items. This is a very desirable characteristic, as it would further reduce the need for ratings of common items. We believe that content-based techniques will fit nicely into the learning framework presented in this paper. Since items correspond to feature vectors, one could extend these feature vectors to contain content-based features. We started to run initial experiments using textual descriptions of movies, extending feature vectors with Boolean features indicating the presence or absence of words. These experiments have not yet led to significant performance improvements. However, we assume that the reason for this is the form of textual movie descriptions available to us for these first experiments, rather than the viability of the method itself.

While the proposed *SVD/ANN* approach leads to performance gains, it is significantly more computationally expensive than the other approaches discussed here. The SVD implementation used in our experiments is a single-vector Lanczos method which is part of the publicly available software package *SVDPACKC* (Berry, 1992). Its computational complexity is $O(3Dz)$, where z is the number of non-zero elements in the matrix and D is the number of dimensions to be computed. In our experiments we observed training times (SVD + network training) ranging from 0.4 seconds for 10 training examples to 2.3 seconds for 50 training examples². While these times would allow for the application of the algorithm as part of an intelligent information agent operating under real-time conditions, we need to keep in mind that we restricted our experiments to 2000 users. Including more users leads to larger matrices to be decomposed and the algorithm will slow down. Therefore, it remains to be seen if similar techniques could be applied to collaborative-filtering services that have accumulated large amounts of data and need to compute predictions under real-time conditions. However, note that the SVD would not have to be recomputed for each user. The SVD of large portions of the available data could be precomputed, and new items that were not part of this analysis could be scaled into the k -dimensional space as described in Section 3.3. The viability, performance and complexity of this approach will be the subject of future research.

² Measured on a 200Mhz Pentium Pro system.

6 SUMMARY AND CONCLUSIONS

In this paper we have identified the shortcomings of correlation-based collaborative filtering techniques and shown how these problems can be addressed through the application of classification algorithms. We believe that the contributions of this paper are twofold. First, we have presented a representation for collaborative filtering tasks that allows the use of virtually any machine learning algorithm. We hope that this will pave the way for further analysis of the suitability of learning algorithms for this task. Second, we have shown that exploiting latent structure in matrices of user ratings can lead to improved predictive performance. In a set of experiments with a database of ratings for motion pictures, we used the singular value decomposition to project user ratings and rated items into a lower dimensional space. This allows users to become predictors for one another's preferences even without any overlap of rated items. Since our society is already being characterized as an information society that suffers from steadily increasing information overload, we regard the automated induction of personalized information filters as an important research problem. The Internet opens up new possibilities to collect enormous amounts of information about users' likes and dislikes. We hope this paper will help develop new ideas for more effective use of this information.

Acknowledgements

We would like to thank the System Research Center of Digital Equipment Corporation for making the *Each-Movie* database available for research.

References

- Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. New Jersey: Princeton University Press.
- Berry, M. W. (1992). Large scale singular value computations. *International Journal of Supercomputer Applications*, 6(1), 13-49.
- Berry, M. W., Dumais, S. T., and O'Brien, G.W. (1995). "Using linear algebra for intelligent information retrieval." *SIAM Review*, 37(4), 1995, 573-595.
- Deerwester, D., Dumais, S. T., Landauer, T. K., Furnas, G.W., and Harshman, R.A. (1990). "Indexing by latent semantic analysis." *Journal of the Society for Information Science*, 41(6), 391-407.
- Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and Evaluating Choices in a Virtual Community of Use. In Proceedings of the Conference on Human Factors in Computing Systems (CHI95), 194-201, Denver, CO, ACM Press.
- Lewis, D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 3-12, London, Springer-Verlag.
- McJones, P. (1997). EachMovie collaborative filtering data set. DEC Systems Research Center. <http://www.research.digital.com/SRC/eachmovie/>.
- Pazzani M., and Billsus, D. (1997). Learning and Revising User Profiles: The identification of interesting web sites. *Machine Learning* 27, 313-331.
- Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1:81-106.
- Resnick, P., Neophytos, I., Mitesh, S. Bergstrom, P. and Riedl, J. (1994) GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of CSCW94: Conference on Computer Supported Cooperative Work, 175-186, Chapel Hill, Addison-Wesley.
- Rumelhart, D. E. and McClelland, J. L. (eds) (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. Cambridge, MA: The MIT Press.
- Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press.
- Shardanand, U. and Maes, P. Social Information Filtering: Algorithms for Automating 'Word of Mouth', In Proceedings of the Conference on Human Factors in Computing Systems (CHI95), 210-217, Denver, CO, ACM Press.

Top-down induction of clustering trees

Hendrik Blockeel

Luc De Raedt

Jan Ramon*

Katholieke Universiteit Leuven, Department of Computer Science
 Celestijnenlaan 200A, B-3001 Heverlee, Belgium
 {Hendrik.Blockeel,Luc.DeRaedt,Jan.Ramon}@cs.kuleuven.ac.be

Abstract

An approach to clustering is presented that adapts the basic top-down induction of decision trees method towards clustering. To this aim, it employs the principles of instance based learning. The resulting methodology is implemented in the TIC (Top down Induction of Clustering trees) system for first order clustering. The TIC system employs the first order logical decision tree representation of the inductive logic programming system TILDE. Various experiments with TIC are presented, in both propositional and relational domains.

1 INTRODUCTION

Decision trees are usually regarded as representing theories for classification. The leaves of the tree contain the classes and the branches from the root to a leaf contain sufficient conditions for classification.

A different viewpoint is taken in *Elements of Machine Learning* [Langley, 1996]. According to Langley, each node of a tree corresponds to a concept or a cluster, and the tree as a whole thus represents a kind of taxonomy or a hierarchy. Such taxonomies are not only output by decision tree algorithms but typically also by clustering algorithms such as e.g. COBWEB [Fisher, 1987]. Therefore, Langley views both clustering and concept-learning as instantiations of the same general technique, the induction of concept hierarchies. The similarity between classification trees and clustering trees has also been noted by Fisher, who points to the possibility of using TDIDT (or TDIDT heuristics)

in the clustering context [Fisher, 1993] and mentions a few clustering systems that work in a TDIDT-like fashion [Fisher and Langley, 1985].

Following these views we study top-down induction of clustering trees. A clustering tree is a decision tree where the leaves do not contain classes and where each node as well as each leaf corresponds to a cluster. To induce clustering trees, we employ principles from instance based learning and decision tree induction. More specifically, we assume that a distance measure is given that computes the distance between two examples. Furthermore, in order to compute the distance between two clusters (i.e. sets of examples), we employ a function that computes a prototype of a set examples. A prototype is then regarded as an example, which allows to define the distance between two clusters as the distance between their prototypes. Given a distance measure for clusters and the view that each node of a tree corresponds to a cluster, the decision tree algorithm is then adapted to select in each node the test that will maximize the distance between the resulting clusters in its subnodes.

Depending on the examples and the distance measure employed one can distinguish two modes. In *supervised* learning (as in the classical top-down induction of decision trees paradigm), the distance measure only takes into account the class information of each example (see e.g. C4.5 [Quinlan, 1993], CART [Breiman *et al.*, 1984]). Also, regression trees (SRT [Kramer, 1996], CART) should be considered supervised learning. In *unsupervised* learning, the examples may not be classified and the distance measure does not take into account any class information. Rather, all attributes or features of the examples are taken into account in the distance measure.

The Top-down Induction of Clustering trees approach is implemented in the TIC system. TIC is a first order

* The authors are listed in alphabetical order.

clustering system as it does not employ the classical attribute value representation but that of first order logical decision trees as in SRT [Kramer, 1996] and TILDE [Blockeel and De Raedt, 1998]. So, the clusters corresponding to the tree will have first order definitions. On the other hand, in the current implementation of TIC we only employ propositional distance measures.

Using TIC we report on a number of experiments. These experiments demonstrate the power of top-down induction of clustering trees. More specifically, we show that TIC can be used for clustering, for regression, and for learning classifiers.

This paper significantly expands on an earlier extended abstract [De Raedt and Blockeel, 1997] in that TIC now contains a pruning method and also that this paper provides new experimental evidence.

This paper is structured as follows. In Section 2 we discuss the representation of the data and the induced theories. Section 3 identifies possible applications of clustering. The TIC system is presented in Section 4. In Section 5 we empirically evaluate TIC for the proposed applications. Section 6 presents conclusions and related work.

2 THE LEARNING PROBLEM

2.1 REPRESENTING EXAMPLES

We employ the *learning from interpretations* setting for inductive logic programming. For the purposes of this paper, it is sufficient to regard each example as a small relational database, i.e. as a set of facts. Within learning from interpretations, one may also specify background knowledge in the form of a Prolog program which can be used to derive additional features of the examples.¹ See [De Raedt and Džeroski, 1994; De Raedt, 1996; De Raedt *et al.*, 1998] for more details on learning from interpretations.

For instance, examples for the well-known mutagenesis problem [Srinivasan *et al.*, 1996] can be described by interpretations. Here, an interpretation is simply an enumeration of all the facts we know about one single molecule: its class, *lumo* and *logp* values, the atoms and bonds occurring in it, certain high-level structures... We can represent it e.g. as follows: {logmutag(-0.7), neg, lumo(-3.025), logp(2.29), atom(d189_1,c,22,-0.11),

¹The interpretation corresponding to each example e is then the minimal Herbrand model of $B \wedge e$.

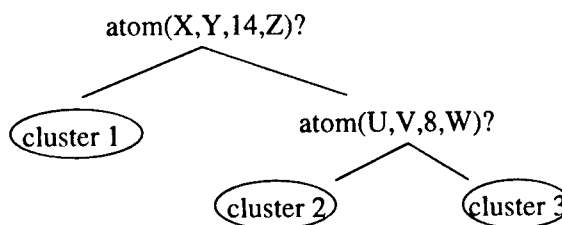


Figure 1: A clustering tree

atom(d189_2,c,22,-0.11), bond(d189_1,d189_2,7),
bond(d189_2,d189_3,7), ... }

2.2 FIRST ORDER LOGICAL DECISION TREES

First order logical decision trees are similar to standard decision trees, except that the test in each node is a conjunction of literals instead of an test on an attribute. They are always binary, as the test can only succeed or fail. A detailed discussion of these trees is beyond the scope of this paper but can be found in [Blockeel and De Raedt, 1998]. We will use these trees to represent clustering trees.

An example of a clustering tree, in the mutagenesis context, is shown in Figure 1. Note that in a classical logical decision tree leaves would contain classes. Here, leaves simply contain sets of examples that belong together. Also note that variables occurring in tests are existentially quantified. The root test, for instance, tests whether there occurs an atom of type 14 in the molecule. The whole set of examples is thus divided into two clusters: a cluster of molecules containing an atom 14 and a cluster of molecules not containing any.

This view is in correspondence with Langley's viewpoint that a test in a node is not just a decision criterion, but also a description of the subclusters formed in this node. In [Blockeel and De Raedt, 1998] we show how a logical decision tree can be transformed into an equivalent logic program, which could alternatively be used to sort examples into clusters. The logic program contains invented predicates that correspond to the clusters.

2.3 INSTANCE BASED LEARNING AND DISTANCES

The purpose of conceptual clustering is to obtain clusters such that intra-cluster distance (i.e. the distance between examples belonging to the same cluster) is as small as possible and the inter-cluster distance (i.e.

the distance between examples belonging to different clusters) is as large as possible.

In this paper, we assume that a distance measure d that computes the distance $d(e_1, e_2)$ between examples e_1 and e_2 is given. Furthermore, there is also a need for measuring the distance between different clusters (i.e. between sets of examples). Therefore we will assume as well the existence of a prototype function p that computes the prototype $p(E)$ of a set of examples E . The distance between two clusters C_1 and C_2 is then defined as the distance $d(p(C_1), p(C_2))$ between the prototypes of the clusters. This shows that the prototypes should be considered as (possibly) partial example descriptions. The prototypes should be sufficiently detailed as to allow the computation of the distances.

For instance, the distance could be the Euclidean distance d_1 between the values of one or more numerical attributes, or it could be the distance d_2 as measured by a first order distance measure such as used in RIBL [Emde and Wettschereck, 1996] or KBG [Bisson, 1992] or [Hutchinson, 1997].

Given the distance at the level of the examples, the principles of instance based learning can be used to compute the prototypes. E.g. d_1 would result in a prototype function p_1 that would simply compute the mean for the cluster, whereas d_2 could result in function p_2 that would compute the (possibly reduced) least general generalisation² of the examples in the cluster.

Throughout this paper we employ only propositional distance measures and the prototype functions that correspond to the instance averaging methods along the lines of [Langley, 1996]. However, we stress that - in principle - we could use any distance measure. Notice that although we employ only propositional distance measures, we obtain first order descriptions of the clusters through the representation of first order logical decision trees.

2.4 PROBLEM-SPECIFICATION

By now we are able to formally specify the clustering problem:

Given

- a set of examples E (each example is a set of tuples in a relational database or equivalently, a set of facts in Prolog),
- a background theory B in the form of a Prolog program,
- a distance measure d that computes the distance between two examples or prototypes,
- a prototype function p that computes the prototype of a set of examples,

Find: a first order clustering tree.

Before discussing how this problem can be solved we take a look at possible applications of clustering trees.

3 APPLICATIONS OF CLUSTERING TREES

Following Langley's viewpoint, a system such as C4.5 can be considered a supervised clustering system where the "distance" metric is the class entropy within the clusters : lower class entropy within a cluster means that the examples in that cluster are more similar with respect to their classes. Since C4.5 employs class information, it is a supervised learner.

Clustering can also be done in an unsupervised manner however. When making use of a distance metric to form clusters, this distance metric may or may not use information about the classes of the examples. Even if it does not use class information, clusters may be coherent with respect to the class of the examples in them.

This principle leads to a classification technique that is very robust with respect to missing class information. Indeed, even if only a small percentage of the examples is labelled with a class, one could perform unsupervised clustering, and assign to each leaf in the concept hierarchy the majority class in that leaf. If the leaves are coherent with respect to classes, this method would yield relatively high classification accuracy with a minimum of class information available. This is quite similar in spirit to Emde's method for learning from few classified examples, implemented in the COLA system [Emde, 1994].

A similar reasoning can be followed for regression, leading to "unsupervised regression"; again this may be useful in the case of partially missing information.

²Using Plotkin's [1970] notion of θ -subsumption or the variants corresponding to structural matching [Bisson, 1992; De Raedt *et al.*, 1997].

We conclude that clustering can extend classification and regression towards unsupervised learning. Another extension in the predictive context is that clusters can be used to predict many or all attributes of an example at once.

Depending on the application one has in mind, measuring the quality of a clustering tree is done in different ways. For classification purposes predictive accuracy on unseen cases is typically used. For regression an often used criterion is the relative error, which is the mean squared error of predictions divided by the mean squared error of a default hypothesis always predicting the mean. This can be extended towards the clustering context if a distance measure and prototype function are available:

$$RE = \frac{\sum_{i=1}^n d(e_i, \hat{e}_i)^2}{\sum_{i=1}^n d(e_i, p)^2}$$

with e_i the examples, \hat{e}_i the predictions and p the prototype. (A prediction is, just like a prototype, a partial example description that is sufficiently detailed to allow the computation of a distance).

If clustering is considered as unsupervised learning of classification or regression trees, the relative error of only the predicted variable or the accuracy with which the class variable can be predicted is a suitable quality criterion. In this case classes should be available for the evaluation of the clustering tree, though not during (unsupervised) learning. Such an evaluation is often done for clusters, see e.g. [Fisher, 1987].

4 TIC: TOP-DOWN INDUCTION OF CLUSTERING TREES

A system for top-down induction of clustering trees called TIC has been implemented as a subsystem of the ILP system TILDE [Blokeel and De Raedt, 1998]. TIC employs the basic TDIDT framework as it is also incorporated in the TILDE system. The main point where TIC and TILDE differ from the propositional TDIDT algorithm is in the computation of the (first order) tests to be placed in a node, see [Blokeel and De Raedt, 1998] for details. Furthermore, TIC differs from TILDE in that it uses other heuristics for splitting nodes, an alternative stopping criterion and alternative tree post-pruning methods. We discuss these topics below.

4.1 SPLITTING

The splitting criterion used in TIC works as follows. Given a cluster C and a test T that will result in two disjoint subclusters C_1 and C_2 of C , TIC computes the distance $d(p(C_1), p(C_2))$, where p is the prototype function. The best test T is then the one that maximizes this distance. This reflects the principle that the inter-cluster distance should be as large as possible.

If the prototype is simply the mean, then maximizing inter-cluster distances corresponds to minimizing intra-cluster distances, and splitting heuristics such as information gain [Quinlan, 1993] or Gini index [Breiman *et al.*, 1984] can be seen as special cases of the above principle, as they minimize intra-cluster class diversity. In the regression context, minimizing intra-cluster variance (e.g. [Kramer, 1996]) is another instance of this principle.

Note that our distance-based approach has the advantage of being applicable to both numeric and symbolic data, and thus generalises over regression and classification.

4.2 STOPPING CRITERIA

Stopping criteria are often based on significance tests. In the classification context a χ^2 -test is often used to check whether the class distributions in the subtrees differ significantly [Clark and Niblett, 1989; De Raedt and Van Laer, 1995]. Since regression and clustering use variance as a heuristic for choosing the best split, a reasonable heuristic for the stopping criterion seems to be the F-test. If a set of examples is split into two subsets, the variance should decrease significantly, i.e.

$$F = \frac{SS/(n-1)}{(SS_L + SS_R)/(n-2)}$$

should be significantly large (SS is the sum of squared differences from the mean inside the set of examples, SS_L and SS_R is the same for the two created subsets of the examples, n is the total number of examples).³

4.3 PRUNING USING A VALIDATION SET

The principle of using a validation set to prune trees is very simple. After using the training set to build a

³The F-test is only theoretically correct for normally distributed populations. Since this assumption may not hold, it should here be considered a *heuristic* for deciding when to stop growing a branch, not a real statistical test.

tree, the quality of the tree is computed on the validation set (predictive accuracy for classification trees, inverse of relative error for regression or clustering trees). For each node of the tree the quality of the tree if it were pruned at that node Q' is compared with the quality Q of the unpruned tree. If $Q' > Q$ then the tree is pruned.

Such a strategy has been successfully followed in the context of classification and regression (e.g. CART [Breiman *et al.*, 1984]) as well as clustering (e.g. [Fisher, 1996]). Fisher's method is more complex than ours in that for each individual variable a different subset of the original tree will be used for prediction.

In the current implementation of TILDE validation set based pruning is available for all settings. For clustering and regression it is the only pruning criterion that is implemented. It is only reliable for reasonably large data sets though. When learning from small data sets performance decreases because the training set becomes even smaller and with a small validation set a lot of pruning is due to random influences.

5 EXPERIMENTS

5.1 DATA SETS

We used the following data sets for our experiments:

- **Soybeans:** this database [Michalski and Chilausky, 1980] contains descriptions of diseased soybean plants. Every plant is described by 35 attributes. A small data set (46 examples, 4 classes) and a large one (307 examples, 19 classes) are available at the UCI repository [Merz and Murphy, 1996].
- **Iris:** a simple database of descriptions of iris plants, available at the UCI repository. It contains 3 classes of 50 examples each. There are 4 numerical attributes.
- **Mutagenesis:** this database [Srinivasan *et al.*, 1996] contains descriptions of molecules for which the mutagenic activity has to be predicted. Originally mutagenicity was measured by a real number, but in most experiments with ILP systems this has been discretized into two values (positive and negative). The database is available at the ILP repository [Kazakov *et al.*, 1996].

Srinivasan *et al.* [1995] introduce four levels of background knowledge; the first 2 contain only structural information (atoms and bonds in the

molecules), the other 2 contain higher level information (attributes describing the molecule as a whole and higher level submolecular structures). For our experiments the tests allowed in the trees can make use of structural information only (Background 2), though for the heuristics numerical information from background 3 can be used.

- **Biodegradability:** a set of 62 molecules of which structural descriptions and molecular weights are given. The biodegradability of the molecules is to be predicted. This is a real number, but has been discretized into four values (fast, moderate, slow, resistant) in most past experiments. The dataset was provided to us by S. Džeroski but is not yet in the public domain.

The data sets were deliberately chosen to include both propositional and relational data sets. For each individual experiment the most suitable data sets were chosen (w.r.t. size, suitability for a specific task, and relevant results published in the literature).

Distances were always computed from all numerical attributes, except when stated otherwise. For the Soybeans data sets all nominal attributes were converted into numbers first.

5.2 EXPERIMENT 1: PRUNING

In this first experiment we want to evaluate the effect of pruning in TIC on both predictive accuracy and tree complexity. We have applied TIC to two databases: Soybeans (large version) and Mutagenesis. We chose these two because they are relatively large (as noted before, the pruning strategy is prone to random influences when used with small datasets).

For both data sets tenfold crossvalidations were performed. In each run the algorithm divides the learning set in a training set and a validation set. Clustering trees are built and pruned in an unsupervised manner. The clustering hierarchy before and after pruning is evaluated by predicting the class of each test example.

In Figure 2, the average accuracy of the clustering hierarchies before and after pruning is plotted against the size of the validation set (this size is a parameter of TIC), and the same is done for the tree complexity. The same results for the Mutagenesis database are summarised in Figure 3.

From the Soybeans experiment it can be concluded that TIC's pruning method results in a slight decrease in accuracy but a large decrease in the number of

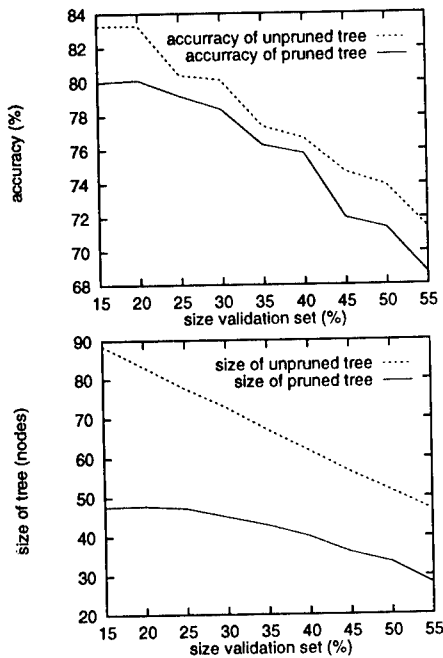


Figure 2: Soybeans: a) Accuracy before and after pruning; b) number of nodes before and after pruning

nodes. The pruning strategy seems relatively stable w.r.t. the size of the validation set. The Mutagenesis experiment confirms these findings (though the decrease in accuracy is less clear here).

5.3 EXPERIMENT 2: COMPARISON WITH OTHER LEARNERS

In this experiment we compare TIC with propositional clustering systems and with classification and regression systems. A comparison with propositional clustering systems is hard to make because few quantitative results are available in the literature, therefore we also compare with supervised learners.

We applied TIC to the Soybean (small) and Iris databases, performing tenfold crossvalidations. Learning is unsupervised, but classes are assumed to be known at evaluation time (the class of a test example is compared with the majority class of the leaf the example is sorted into). Table 1 compares the results with those obtained with the supervised learner TILDE.

We see that TIC obtains high accuracies for these problems. The only clustering result we know of is for COBWEB, which obtained 100% on the Soybean data set. This difference is not significant. TILDE's ac-

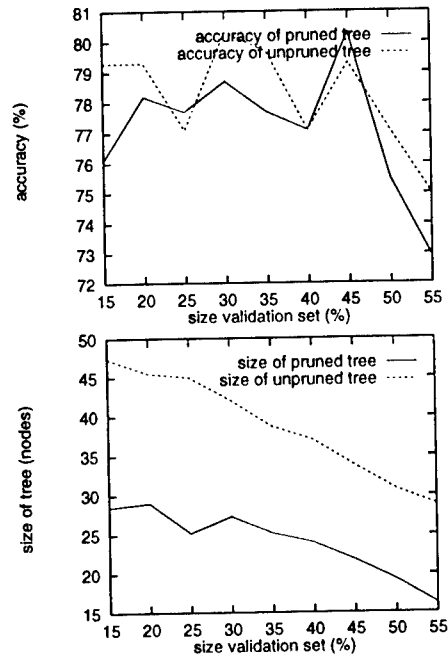


Figure 3: Mutagenesis: Accuracy and size of the clustering trees

Database	TIC		TILDE	
	acc.	tree size	acc.	tree size
Soybean	97%	3.9 nodes	100%	3 nodes
Iris	92%	15 nodes	94%	4 nodes

Table 1: Comparison of TIC with a supervised learner (averages over 10-fold crossvalidation).

curacies don't differ much from those of TIC which induced the hierarchy without knowledge of the classes. Tree sizes are smaller though.

We have also performed an experiment on the Biodegradability data set, predicting numbers. For this dataset the F-test stopping criterion was used (significance level 0.01), but no validation set was used given the small size of the data set. The distance used is the difference between class values. Table 2 compares TIC's performance with TILDE's (classification, leave-one-out) and SRT's (regression, sixfold).

Our conclusions are that a) for unsupervised learning TIC performs almost as well as other unsupervised or supervised learners, if classification accuracy is measured; and b) while there is clearly room for improvement with respect to using TIC for regression, post-discretization of the regression predictions shows that this approach is competitive with classical approaches to classification.

l.o.o. TILDE	classification	acc. = 0.532
l.o.o. TIC	regression	RE = 0.740
l.o.o. TIC	classif. via regression	acc. = 0.565
6-fold SRT	regression	RE = 0.34
6-fold TIC	regression	RE = 1.13

Table 2: Comparison of regression and classification on the biodegradability data (l.o.o.=leave-one-out).

5.4 EXPERIMENT 3: PREDICTING MULTIPLE ATTRIBUTES

Clustering allows to predict multiple attributes. Since examples in a leaf must resemble each other as much as possible, attributes must also agree as much as possible.

By sorting unseen examples down a cluster tree and comparing all attributes of the example with the prototype attributes, we get an idea of how good the tree is. This is an extension of the classical evaluation, as each attribute in turn is a class now.

We did a tenfold crossvalidation for the following experiment: using the training set a clustering tree is induced. Then, all examples of the test set are sorted in this hierarchy, and the prediction for all of their attributes is evaluated. For each attribute, the value that occurs most frequently in a leaf is predicted for all test examples sorted in that leaf.

We used the large soybean database, with pruning. Table 3 summarizes the accuracies obtained for each attribute and compares with the accuracy of majority prediction. The high accuracies show that most attributes can be predicted very well, which means the clusters are very coherent. The mean accuracy of 81.6% does not differ significantly from the $83 \pm 2\%$ reported in [Fisher, 1996].

5.5 EXPERIMENT 4: HANDLING MISSING INFORMATION

It can be expected that clustering, making use of more attributes than just class attributes, is more robust with respect to missing values. We showed in Experiment 2 that unsupervised learners (where the heuristics do not use any class information at all) can yield trees with predictive accuracies close to those of supervised learners, but all class information was still available for assigning classes to leaves after the tree was built.

In this experiment, we measure the predictive accu-

name	range	default	acc.
date	0-6	21.2%	46.3%
plant_stand	0-1	52.1%	85.0%
precip	0-2	68.4%	79.2%
temp	0-2	58.3%	75.6%
hail	0-1	68.7%	71.3%
crop_hist	0-3	32.2%	45.0%
area_damaged	0-3	32.9%	54.4%
severity	0-2	49.2%	63.2%
seed_tmt	0-2	45.6%	51.1%
germination	0-2	32.2%	45.0%
plant_growth	0-1	65.8%	96.4%
leaves	0-1	89.3%	96.4%
leafspots_halo	0-2	49.5%	85.3%
leafspots_marg	0-2	52.2%	86.6%
leafspots_size	0-2	47.8%	87.0%
leaf_shread	0-1	75.9%	81.4%
leaf_malf	0-1	87.3%	88.3%
leaf_mild	0-2	83.7%	88.9%
stem	0-1	54.1%	98.4%
lodging	0-1	80.7%	80.0%
stem_cankers	0-3	58.3%	90.6%
canker_lesion	0-3	49.1%	88.9%
fruiting_bodies	0-1	73.6%	84.3%
external_decay	0-2	75.6%	91.5%
mycelium	0-1	95.8%	96.1%
int_discolor	0-2	86.6%	95.4%
sclerotia	0-1	93.2%	96.1%
fruit_pods	0-3	62.7%	91.2%
fruit_spots	0-4	53.4%	87.0%
seed	0-1	73.9%	85.7%
mold_growth	0-1	80.5%	86.6%
seed_discolor	0-1	79.5%	84.0%
seed_size	0-1	81.8%	88.6%
shriveling	0-1	83.4%	87.9%
roots	0-2	84.7%	95.8%
mean			81.6%

Table 3: Prediction of all attributes together in the Soybean data set

racy of trees when class information as well as other information may be missing, not only for learning, but also for assigning classes to leaves afterwards, and this for several levels of missing information. Our aim is to investigate how predictive accuracy deteriorates with missing information, and to compare clustering systems that use only class information with systems that use more information.

We have used the Mutagenesis data set for this experiment (for each example, there was a fixed probability that the value of a certain attribute was removed from the data; this probability was increased for consecutive experiments), comparing the use of only class information (*logmutag*) with the use of three numerical variables (among which the class) for computing

available numerical data	logmutag	all three
100%	0.80	0.81
50%	0.78	0.79
25%	0.72	0.77
10%	0.67	0.74

Table 4: Classification accuracies obtained for Muta-genesis with several distance functions, and on several levels of missing information.

distances. This experiment is similar in spirits to the ones performed with COLA [Emde, 1994]. Table 4 shows the results. As expected, performance degrades less quickly when more information is available, which supports the claim that the use of more than just class information can improve performance in the presence of missing information.

6 CONCLUSIONS AND RELATED WORK

We have presented a novel first order clustering system TIC within the TDIDT class of algorithms. TIC integrates ideas from concept-learning (TDIDT), from instance based learning (the distances and the prototypes), and from inductive logic programming (the representations) to obtain a clustering system. Several experiments were performed that illustrate the type of tasks TIC is useful for.

As far as related work is concerned, our work is related to KBG [Bisson, 1992], which also performs first order clustering. In contrast to the current version of TIC, KBG does use a first order similarity measure, which could also be used within TIC. Furthermore, KBG is an agglomerative (bottom-up) clustering algorithm and TIC a divisive one (top-down). The divisive nature of TIC makes TIC as efficient as classical TDIDT algorithms. A final difference with KBG is that TIC directly obtains logical descriptions of the clusters through the use of the logical decision tree format. For KBG, these descriptions have to be derived in a separate step because the clustering process only produces the clusters (i.e. sets of examples) and not their description.

The instance-based learner RIBL [Emde and Wettschereck, 1996] uses an advanced first order distance metric that might be a good candidate for incorporation in TIC.

While [Fisher, 1993] first made the link between TDIDT and clustering, our work is inspired mainly by [Langley, 1996]. From this point of view, our work

is closely related to SRT [Kramer, 1996], who builds regression trees in a supervised manner. TIC can be considered a generalization of SRT in that TIC can also build trees in an unsupervised manner, and can predict multiple values. Finally, we should also refer to a number of other approaches to first order clustering, which include Kluster [Kietz and Morik, 1994], [Yoo and Fisher, 1991], [Thompson and Langley, 1991] and [Ketterlin *et al.*, 1995].

Future work on TIC includes extending the system so that it can employ first order distance measures, and investigating the limitations of this approach (which will require further experiments).

Acknowledgements

Hendrik Blockeel is supported by the Flemish Institute for the Promotion of Scientific and Technological Research in Industry (IWT). Luc De Raedt is supported by the Fund for Scientific Research of Flanders.

This work is part of the European Community Esprit project no. 20237, Inductive Logic Programming 2. The authors thank Stefan Kramer, who performed the SRT experiments, Sašo Džeroski, who provided the Biodegradability database, Luc Dehaspe and Kurt Driessens for proofreading the paper, and the anonymous referees for their very valuable comments.

References

- [Bisson, 1992] G. Bisson. Conceptual clustering in a first order logic representation. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 458–462. John Wiley & Sons, 1992.
- [Blockeel and De Raedt, 1998] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 1998. To appear.
- [Breiman *et al.*, 1984] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
- [Clark and Niblett, 1989] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–284, 1989.
- [De Raedt and Blockeel, 1997] L. De Raedt and H. Blockeel. Using logical decision trees for clustering. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 133–141. Springer-Verlag, 1997.
- [De Raedt and Džeroski, 1994] L. De Raedt and S. Džeroski. First order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

- [De Raedt and Van Laer, 1995] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Workshop on Algorithmic Learning Theory*, volume 997 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.
- [De Raedt et al., 1997] L. De Raedt, P. Idestam-Almqvist, and G. Sablon. θ -subsumption for structural matching. In *Proceedings of the 9th European Conference on Machine Learning*, pages 73–84. Springer-Verlag, 1997.
- [De Raedt et al., 1998] L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. Three companions for first order data mining. In N. Lavrač and S. Džeroski, editors, *Inductive Logic Programming for Knowledge Discovery in Databases*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1998. To appear.
- [De Raedt, 1996] L. De Raedt. Induction in logic. In R.S. Michalski and Wnek J., editors, *Proceedings of the 3rd International Workshop on Multistrategy Learning*, pages 29–38, 1996.
- [Emde and Wettschereck, 1996] W. Emde and D. Wettschereck. Relational instance-based learning. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.
- [Emde, 1994] W. Emde. Inductive learning of characteristic concept descriptions. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 51–70, Sankt Augustin, Germany, 1994. Gesellschaft für Mathematik und Datenverarbeitung MBH.
- [Fisher and Langley, 1985] D. Fisher and P. Langley. Approaches to conceptual clustering. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 691–697, Los Altos, CA, 1985. Morgan Kaufmann.
- [Fisher, 1987] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.
- [Fisher, 1993] D. H. Fisher. Database management and analysis tools of machine induction. *Journal of Intelligent Information Systems*, 2, 1993.
- [Fisher, 1996] D. H. Fisher. Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research*, 4:147–179, 1996.
- [Hutchinson, 1997] A. Hutchinson. Metrics on terms and clauses. In *Proceedings of the 9th European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence, pages 138–145. Springer-Verlag, 1997.
- [Kazakov et al., 1996] D. Kazakov, L. Popelinsky, and O. Stepankova. ILP datasets page [<http://www.gmd.de/ml-archive/datasets/%ilp-res.html>], 1996.
- [Ketterlin et al., 1995] A. Ketterlin, P. Gancarski, and J.J. Korczak. Conceptual clustering in structured databases : a practical approach. In *Proceedings of KDD-95*, 1995.
- [Kietz and Morik, 1994] J.U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14:193–217, 1994.
- [Kramer, 1996] S. Kramer. Structural regression trees. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, 1996.
- [Langley, 1996] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.
- [Merz and Murphy, 1996] C.J. Merz and P.M. Murphy. UCI repository of machine learning databases [<http://www.ics.uci.edu/~mlearn/mlrepository.html>], 1996. Irvine, CA: University of California, Department of Information and Computer Science.
- [Michalski and Chilausky, 1980] R.S. Michalski and R.L. Chilausky. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Policy analysis and information systems*, 4, 1980.
- [Plotkin, 1970] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.
- [Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann, 1993.
- [Srinivasan et al., 1995] A. Srinivasan, S.H. Muggleton, and R.D. King. Comparing the use of background knowledge by inductive logic programming systems. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, 1995.
- [Srinivasan et al., 1996] A. Srinivasan, S.H. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85, 1996.
- [Thompson and Langley, 1991] K. Thompson and P. Langley. Concept formation in structured domains. In D. Fisher, M. Pazzani, and P. Langley, editors, *Concept formation: knowledge and experience in unsupervised learning*. Morgan Kaufmann, 1991.
- [Yoo and Fisher, 1991] J. Yoo and D. Fisher. Concept formation over explanations and problem-solving experience. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 630 – 636. Morgan Kaufmann, 1991.

A Supra-Classifier Architecture for Scalable Knowledge Reuse

Kurt D. Bollacker and Joydeep Ghosh

Department of Electrical and Computer Engineering,
University of Texas at Austin,
Austin, TX 78712
{kdb,ghosh}@pine.ece.utexas.edu

Abstract

When faced with inadequate information, humans often use knowledge gained from previous experience to help them in making decisions. Even when this knowledge is spread thinly among many previous experiences, humans are able to effectively accumulate and apply it to a current classification task of interest. Inspired by human knowledge reuse, we have previously introduced a general framework for the use of knowledge embodied in existing classifiers to aid in a new classification task. In this framework, a *supra-classifier* is built to make decisions based on the outputs of large numbers of previously trained classifiers designed for different, but possibly relevant tasks. In this article, we discuss the *Hamming Nearest Neighbor (HNN)* supra-classifier architecture and mathematically show its usefulness. Experiments on public domain data sets demonstrate the practicality of the framework and HNN supra-

classifier when faced with very few training examples.

Keywords: Knowledge Transfer, Nearest Neighbor, Curse of Dimensionality

1 INTRODUCTION

In this paper we mathematically analyze the Hamming Nearest Neighbor (HNN) supra-classifier architecture for integrating multiple knowledge sources, based on a recently introduced framework for knowledge reuse (Bollacker & Ghosh, 1997). We demonstrate the ability of the HNN supra-classifier to theoretically approach optimal performance even with minimal training samples, if enough relevant knowledge is available. In particular, we show that in the limit of having only one training sample of each target class but with an infinite number of independent, (at least weakly) relevant previously trained classifiers available, a perfect supra-classifier is approached. We first review the motivation for and existing research related to knowledge reuse and summarize our supra-classifier based knowledge reuse framework.

1.1 MOTIVATION

A person is able to quickly and robustly recognize patterns from very few samples. This is due, at least in part, to the use of the vast reservoir of experiential knowledge from which he/she may draw. He/she may use relevant learned knowledge to better understand the problem domain, thus helping to constrain the interpretation of current data. One of the most impressive traits of human knowledge reuse is the ability to draw simultaneously from a large number of previous experiences quickly and easily. Each bit of learned knowledge may not help much, but as a whole, the knowledge gained from the whole of many experiences can paint a very clear picture of the problem domain.

Unlike humans, artificial classification systems often depend greatly on the set of training samples to make classification decisions. If the training set insufficiently represents the "essence" of a classification task, then creation of a well generalizing classifier for that task may not be possible. It is natural then, to suggest that in the construction of artificial classifiers, the inclusion of previously learned knowledge embodied in previously existing classifiers is a potential approach to the problem of inadequate training data.

Also unlike humans, artificial systems have often failed in their ability to use a large number of weakly relevant information sources. For example, the "curse of dimensionality" (e.g. see (Friedman, 1994)) is given this name (at least partially) because of the difficulties it represents for the creators of well performing artificial classifiers when faced with a high dimensional input. An ideal architecture for classifier knowledge reuse would be *scalable* in the sense that it can effectively handle the high di-

mensional input resulting from use of large numbers of previously trained classifiers, even if most of them are only marginally relevant.

1.2 PREVIOUS RESEARCH

The most common approaches to knowledge reuse are ones that are often not considered to be "knowledge reuse" per se, but instead cast previously gained relevant knowledge as a "domain" which is crafted (often in an ad hoc manner) to represent the underlying semantic or physical structure of the problem. For example, Bayesian approaches (e.g. (Mackay, 1995)) reuse knowledge in the form of prior class probabilities and prior distributions assumed for the model parameters, while many classifier architectures use the structure and value of model parameters to represent domain knowledge (e.g. the discriminant function in statistical classifiers (Fukunaga, 1990), size and order of features in decision trees (Mitchell, 1997), and the type and number of hidden units, amount and form of regularization in feed-forward neural networks (Ghosh & Tumer, 1994)). Such approaches can work very well if the inductive bias matches the problem very closely. However, in practice it may be quite difficult to select and tune a proper model. Also, standard assumptions used (independence among variables, Gaussian distributions, etc.) to make the problem tractable often result in a loss of accuracy (Heckerman, 1997; Mackay, 1995).

There has been much work on knowledge intensive learning focusing on symbolic rules extracted from and used in the creation of neural classifiers (e.g. (Towell & Shavlik, 1994; Mahoney & Mooney, 1993)). If knowledge can be represented as rules, then it may be used to build a better classifier. However, most of the approaches cannot reuse knowl-

edge from general classifiers and have not demonstrated scalability to a large number of simultaneous weak information sources.

Some recent work in knowledge reuse has focused on the automated extraction and reuse of knowledge from the data sets of other relevant classifiers, including reuse of the trained classifiers themselves. Under the belief that related classification tasks may benefit from common internal features, Caruana (Caruana, 1995) has created a multilayer perceptron (MLP) based multiple classifier system that is trained *simultaneously* to perform several related classification tasks. Baxter (Baxter, 1994) has developed a rigorous analysis of a similar type of architecture, showing that as the number of simultaneously trained tasks increases, the number of examples needed per task for good generalization decreases. Pratt (Pratt, 1994) has explored a similar knowledge reuse method in which some of the trained weights from one MLP network are used to initialize weights in an MLP to be trained for a later, related task. A different approach is taken by Thrun (Thrun & O'Sullivan, 1996), who proposed a method to estimate classifier relevance by measuring how much better a classifier performs with a reused scaling vector for Nearest Neighbor classifiers. Tasks with mutually helpful scaling vectors can be "clustered" into related groups.

Recently, popular approaches such as stacking, committees, ensembles, and mixture of experts also use multiple classifiers. However, since most these classifiers try to solve the *same* task (though they may specialize in different input regions) and do not use previously created classifiers, they are simply good methods of decomposing a classification task into simpler tasks and do not generally reuse previous knowledge.

2 KNOWLEDGE REUSE FRAMEWORK

In our framework for knowledge reuse (Bollacker & Ghosh, 1997), classifiers previously trained to perform (potentially relevant) classification tasks are termed *support* classifiers as indicated in Figure 1. Support

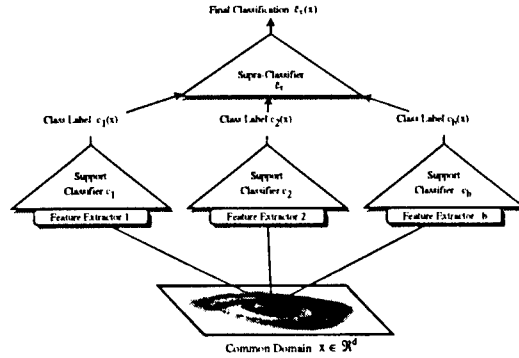


Figure 1: A *Supra-Classifier* Reuse Architecture.

classifiers are generally (but not always) designed for tasks other than the current *target* classification task of interest. Our reuse strategy is to apply the input values of each of the training samples available for the target task to all available classifiers sharing the input domain with the target classifier. The output class labels of the target and support classifiers are observed by a second stage *supra-classifier* which makes the ultimate classification ($\hat{c}_\tau(\cdot)$ in the figure). Since no internal information is being used, the support classifiers can be of any type. All classifiers feeding into the supra-classifier must share an ultimately common input domain. This domain may be broad, such as the domain of all images.

2.1 A FEW DEFINITIONS

Let the target classification task be τ , and let τ have discrete range \mathcal{S}_τ and d di-

mensional input domain space \mathbb{R}^d . Let $\{x, y\}_\tau : x \in \mathbb{R}^d, y \in \mathcal{S}_\tau$ be the set of training examples for task τ . We assume that $\{x, y\}_\tau$ is a sample set from the true distribution for task τ with associated random variable $(X_\tau, Y_\tau) \in (\mathbb{R}^d, \mathcal{S}_\tau)$. Our goal is to find the most likely value of the conditional marginal $Y_\tau | (X_\tau = x)$ and define this maximum likelihood function to be $t(x) = \arg \max_y P(Y_\tau = y | X_\tau = x)$. Thus, $t(\cdot) : t(\cdot) \in \mathcal{S}_\tau$ is the target function that we would like to approximate using the information in $\{x, y\}_\tau$. Let \mathcal{B} be a set of support classification tasks which have the same input domain space \mathbb{R}^d as task τ . Let $\{c_b(\cdot)\} : b \in \mathcal{B}$ be the corresponding set of classifiers where each $c_b(\cdot)$ maps $\mathbb{R}^d \mapsto \mathcal{S}_b : b \in \mathcal{B}$.¹ Let \hat{X}_τ be the random variable associated with the input values of training sample set $\{x, y\}_\tau$. Let $T_\tau : T_\tau = t_\tau(\hat{X}_\tau)$ be defined as the random variable associated with the target function of \hat{X}_τ . Similarly, let $C_b : C_b = c_b(\hat{X}_\tau)$ be the random variables resulting from the application of \hat{X}_τ to the support classifiers.

An *Ideal Supra-Classifier* $c_\tau^*(x)$ will always choose the most likely class of the $y \in \mathcal{S}_\tau$ given the class labels $\{c_b(x)\} : b \in \mathcal{B}$. More specifically, For any given $\{z_b : z_b \in \mathcal{S}_b\} : b \in \mathcal{B}$ we can define the maximum probability function $m(\cdot)$ as $m(\{z_b\} : b \in \mathcal{B}) = \arg \max_y P(T_\tau = y | \{C_b = z_b\} : B \in \mathcal{B})$. We can then define an ideal classifier based on this maximum probability function as

$$c_\tau^*(x) = m(\{c_b(x)\} : b \in \mathcal{B}). \quad (1)$$

where $c_\tau^*(\cdot)$ has associated random variable $C_\tau^* : C_\tau^* = c_\tau^*(\hat{X}_\tau)$. In practice if the number of support classifiers is quite large, Equation 1 is not directly scalable due to the *curse*

¹Although some of the support classifiers may have been trained for task τ directly, in general $b \neq \tau$ and $\mathcal{S}_\tau \neq \mathcal{S}_b$, as the tasks are different.

of dimensionality (Friedman, 1994). Therefore, approximating approaches to Equation 1 are required. An empirical comparison of several such approaches was made in (Bollacker & Ghosh, 1997). Somewhat surprisingly, for the case of few training examples, the simple *Hamming Nearest Neighbor* was seen to provide the best knowledge reuse performance. In this paper we provide a mathematical analysis of the HNN architecture to better understand its excellent performance and scalable properties.

3 HAMMING NEAREST NEIGHBOR (HNN) SUPRA-CLASSIFIER

The HNN classifier is similar to a traditional nearest neighbor which operates in a Euclidean space. The HNN operates in a "Hamming space" where the distance between two discrete values is 0 if they are the same and 1 if different. If $I(\cdot)$ is the indicator function, then the (Hamming) distance measure between two samples x_{train} and x_{test} can be calculated as

$$H(x_{train}, x_{test}) = \sum_{b: b=1 \dots |\mathcal{B}|} I(c_b(x_{train}) \neq c_b(x_{test})).$$

For each test sample, the Hamming Nearest Neighbor (HNN) supra-classifier will choose the class label of the training sample with the smallest Hamming distance from it. We now proceed to analyze the HNN classifier to show that under certain assumptions, as more support classifiers are included, the supra-classifier can approach perfect performance, even if the supporting classifiers are not very relevant to the current task. Proofs of the following lemmas are given in (Bollacker & Ghosh, 1998b).

Definitions and Assumptions

Let $\mathcal{R} \subset \mathbb{R}^d$ be a region of space in which a distribution of samples $\{x\}$ of non-zero density lie. Let $t(x) \in \mathcal{S}_\tau \equiv \{1, \dots, N\}$ be the true classifier label of some sample $x \in \mathcal{R}$ where $\{1, \dots, N\}$ is the finite set of discrete target class labels. Let $c_b(x) \in \mathcal{S}_b \equiv \{1, \dots, M\}$ be a support classifier labeling. Define $P^j : P^j = P(t(x) = j)$, and $P_i^j : P_i^j = P(t(x) = j | c_b(x) = i)$ where x is a sample chosen randomly from \mathcal{R} . P^j can be interpreted to mean the probability of choosing a sample of target class j when picking randomly from \mathcal{R} . P_i^j is the probability of choosing a sample of target class j when picking randomly from the subset of samples in \mathcal{R} which are of support class i . It can be seen that if we have two samples x_α and x_γ , randomly and independently drawn from \mathcal{R} , then the probability of them having the same target class label j is $(P^j)^2$.

Lemma 1:

If the target classes $j : j = 1 \dots N$ have equal prior probabilities P^j , then

$$\forall i, \sum_{j=1}^N (P_i^j)^2 \geq \frac{1}{N}.$$

This result is used to show that knowing that two samples have the same support class label improves their chance of being of the same target class.

Lemma 2:

If x_α and x_γ are drawn randomly and independently from \mathcal{R} , and the target classes $j : j = 1 \dots N$ have equal prior probabilities P^j , then

$$P(t(x_\alpha) = t(x_\gamma) | c_b(x_\alpha) = c_b(x_\gamma)) \geq P(t(x_\alpha) = t(x_\gamma)).$$

The proof consists of noticing that the chance of two random samples being the same target class is minimized when all of the P^j are equal (Lemma 1) and that for

any partitioning of the set of samples in \mathcal{R} induced by the labels $\{i\} : c_b(\cdot) = i, i = 1 \dots M$, the probability of two samples being the same target class cannot be reduced further.

Now we can use Lemma 2 to show that two samples randomly and independently chosen from \mathcal{R} have as good or better chance of being of the same support class if they are of the same target class than if they are of different target classes.

Lemma 3:

If x_α , x_β , and x_γ are drawn randomly and independently from \mathcal{R} , then

$$\begin{aligned} &P(c_b(x_\alpha) = c_b(x_\gamma) | t(x_\alpha) = t(x_\gamma)) \\ &\geq P(c_b(x_\beta) = c_b(x_\gamma) | t(x_\beta) \neq t(x_\gamma)). \end{aligned}$$

The case of equality occurs only when $c_b(\cdot)$ is independent of $t(\cdot)$.

This Lemma is interesting in the context of a Nearest Neighbor classifier. Let x_α and x_β be two training samples and x_γ be a test sample.

Now let us consider the use of n support classifiers to build an HNN supra-classifier. Taking the complements of the events in Lemma 3, $P(c_b(x_\beta) \neq c_b(x_\gamma) | t(x_\beta) \neq t(x_\gamma)) \geq P(c_b(x_\alpha) \neq c_b(x_\gamma) | t(x_\alpha) = t(x_\gamma))$, and summing over all $b : b = 1 \dots n$, we can write

$$\begin{aligned} &\sum_{b=1}^n P(c_b(x_\beta) \neq c_b(x_\gamma) | t(x_\beta) \neq t(x_\gamma)) \geq \\ &\sum_{b=1}^n P(c_b(x_\alpha) \neq c_b(x_\gamma) | t(x_\alpha) = t(x_\gamma)). \end{aligned}$$

If we let $\delta_b \geq 0$ be the difference between each pair of terms in the sums, and let $\delta^n = \sum_{b=1}^n \frac{\delta_b}{n}$, then we can write

$$\sum_{b=1}^n P(c_b(x_\beta) \neq c_b(x_\gamma) | t(x_\beta) \neq t(x_\gamma)) -$$

$$\sum_{b=1}^n P(c_b(x_\alpha) \neq c_b(x_\gamma) | t(x_\alpha) = t(x_\gamma)) = n\delta^n. \quad (2)$$

Note that $\delta_b = 0$ only if $c_b(\cdot)$ is independent of $t(\cdot)$, and thus is of no use for the target task. We will assume, as the number of support classifiers grows, the fraction of useful ones does not approach zero, (i.e. Let $\delta^\infty = \lim_{n \rightarrow \infty} \delta^n > 0$).

Returning to the definition of the HNN classifier, we write the Hamming distances

$$D_n(x_\alpha, x_\gamma) = \sum_{b=1}^n I(c_b(x_\alpha) \neq c_b(x_\gamma) | t(x_\alpha) = t(x_\gamma)) \quad (3)$$

$$D_n(x_\beta, x_\gamma) = \sum_{b=1}^n I(c_b(x_\beta) \neq c_b(x_\gamma) | t(x_\beta) \neq t(x_\gamma)) \quad (4)$$

Theorem 1:

If the support classifiers $c_b(\cdot)$ are independent of each other conditionally on the target class $t(\cdot)$, $t(x_\alpha) = t(x_\gamma) \neq t(x_\beta)$, and the priors (P^j) for each target class are equal, then

$$\lim_{n \rightarrow \infty} P(D_n(x_\beta, x_\gamma) > D_n(x_\alpha, x_\gamma)) = 1.$$

Proof:

We will apply the weak law of large numbers as given in (Billingsley, 1979), which states $\lim_{n \rightarrow \infty} P(|\frac{\sum_{i=1}^n Y_i - \sum_{i=1}^n E[Y_i]}{n}| \geq \epsilon) = 0$ for independent trials Y_i and all $\epsilon > 0$. Noticing that $P(c_b(x_1) \neq c_b(x_2)) = E[I(c_b(x_1) \neq c_b(x_2))]$ where $I(\cdot)$ is the indicator function and substituting from Equations 2, 3, and 4, we can write

$$\lim_{n \rightarrow \infty} P(|\frac{D_n(x_\beta, x_\gamma) - D_n(x_\alpha, x_\gamma) - n\delta^n}{n}| \geq \epsilon) = 0,$$

which leads to

$$\equiv \lim_{n \rightarrow \infty} P(|\frac{D_n(x_\beta, x_\gamma) - D_n(x_\alpha, x_\gamma) - n\delta^n}{n}| < \epsilon) = 1.$$

$$\Rightarrow \lim_{n \rightarrow \infty} P(\frac{D_n(x_\beta, x_\gamma) - D_n(x_\alpha, x_\gamma)}{n} > \delta^n - \epsilon) = 1,$$

Since we assume $\delta^\infty > 0$, we can choose a sufficiently small $\epsilon : \epsilon < \delta^\infty$ and write

$$\equiv \lim_{n \rightarrow \infty} P(\sum_{b=1}^n D_n(x_\beta, x_\gamma) > D_n(x_\alpha, x_\gamma)) = 1. \quad \square$$

Theorem 1 states that in the limit of an infinite number of conditionally independent and (at least barely useful) support classifiers being available, the probability that the HNN classifier will predict the true target class approaches 1. It should also be noted that Theorem 1 holds even if there is only one training sample of each target class. This results leads to the observation that under certain conditions, a wealth of features can compensate for a dearth of samples. This is counter to the conventional wisdom that more feature usually requires more training samples. The trick is that we are not working in a Euclidian feature space, and so do not fall victim so easily to the typical curse of dimensionality problems. Despite this compelling analysis of the HNN classifier, a careful separation of theory from practice should be made. While a perfect classifier is theoretically possible, in general it would be impossible to gather an infinite number of independent, relevant support classifiers. Also, since the HNN supra-classifier is computationally linear in the number support classifiers, an infinite number could generally never be used. However, the results lead us

to believe that as more independent support classifiers or training samples are included, the better HNN will perform. This is supported by empirical evidence (Bollacker & Ghosh, 1998a). The question of how fast the HNN classifier approaches perfect performance is only answerable for a specific set of training samples and included support classifiers. An analysis of large deviations as in (Billingsley, 1979) suggests that HNN would approach its limit exponentially fast as a function of the relevance of the support classifiers.

4 EXPERIMENTS

Previously, we have explored the empirical performance of the HNN supra-classifier (Bollacker & Ghosh, 1997), with some of the results discussed here. We used three public domain data sets from the U.C Irvine Machine Learning database and partitioned the samples from each data set into two disjoint and unequal sized subsets *based on their class labels*. The larger subset was used to create several two-class (non-target) problems using all combinations of two classes not to be used as target classes. First, a 20000 sample capital English letter data set (LR) was divided into the target data set consisting of the five classes "H", "L", "O", "R", and "S", and 210 other classifier data sets consisting of two-class combinations of the other 21 classes. Second, a spoken vowel data set (VOW) consisted of 990 samples evenly distributed among 11 spoken vowels. The two classes "hud" and "hed" were chosen to form the target classifier task and the remaining 9 classes were used to construct 36 other 2-class classification tasks in a manner similar to the LR data set. Third, the well known soybean data set (SOY) consisting of 683 samples. The three classes "phytophthora-rot", "brown-spot", and "al-

ternaria leaf-spot" were chosen to be the target classes, and the remaining 16 classes were used to generate 120 other (2-class) classifiers.

The three data sets were randomly partitioned into equal sized training and test sets. The target training set was used to create MLP and single nearest neighbor (1-NN) classifiers for each target problem. The 210 LR 2-class classifiers were trained MLP's, while the 120 soy and 36 VOW other 2-class classifiers were single Nearest Neighbor (1-NN) classifiers. These classifier architectures were chosen for their good performance on those tasks. In order to consider the case of few available target training samples, only a fraction of the available target training samples was actually used. The set of support classifiers for each problem consisted of simple classifiers for the target class and all of the 2-class classifiers built using non-target class samples. Target training sets over a range of sizes were applied to the support classifiers for the three problems. The outputs of these support classifiers were then used as the input vector for an HNN supra-classifier. Results using the LR data set (averaged over 20 trials), SOY data set (100 trials), and VOW data set (100 trials) can be seen in Figures 2, 3, and 4 respectively. For all three data sets, the HNN supra-classifier showed improved performance over all of the unaided classifiers, especially with small target training sets. Moreover, the difference between the HNN and unaided 1-NN for few examples was calculated to be statistically significant with greater than a 99% certainty.

5 CONCLUSIONS AND FUTURE WORK

We have discussed the motivation for reuse of knowledge from previously trained

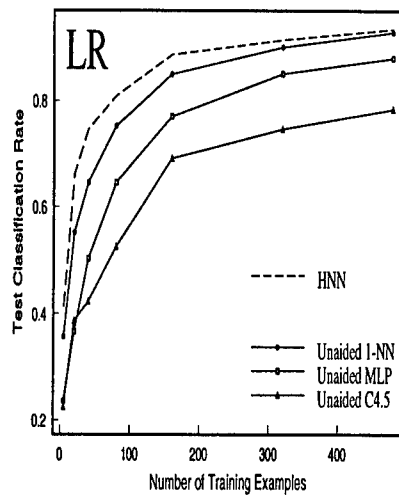


Figure 2: Test rate vs. number of training samples on the letter recognition data set.

classifiers and presented a framework for such reuse which includes the concept of supra-classifiers. We introduce the Hamming Nearest Neighbor supra-classifier and demonstrate its usefulness both analytically and empirically. This gives evidence that the HNN supra-classifier architecture would be a useful approach to the problems of inadequate training samples.

In the future, we intend to do further analysis of the HNN supra-classifier to determine the convergence rate as more support classifiers and target training samples are added. A practical extension will be an application to a truly complex problem domain. We envision the eventual construction of a "warehouse" of previously constructed reusable classifiers for a large domain of interest (e.g. image databases), where the set of support classifiers will serve as an efficient representation of the problem domain knowledge.

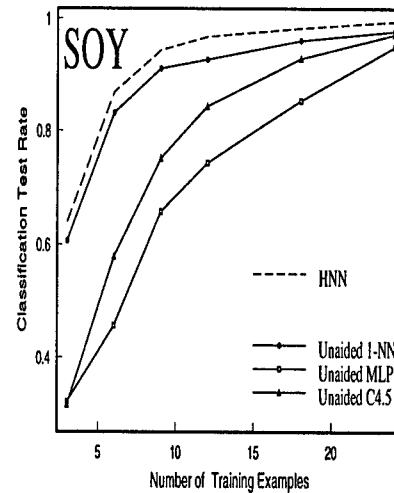


Figure 3: Test rate vs. number of training samples on the soybean data set.

Acknowledgements

This work was supported by the Army Research Office contracts DAAH-94-G-0417 and DAAH 049510494, Texas ATP grant #442, and the National Science Foundation.

References

- Baxter, J. (1994). *Learning Internal Representations*. Ph.D. thesis, The Flinders University of South Australia.
- Billingsley, P. (1979). *Probability and Measure*. John Wiley and Sons, New York.
- Bollacker, K. D., & Ghosh, J. (1997). Knowledge reuse in multiple classifier systems. *Pattern Recognition Letters*, 18(11-13), 1385-1390.
- Bollacker, K. D., & Ghosh, J. (1998a). On the design of supra-classifiers for knowledge reuse. In *Proceedings of*

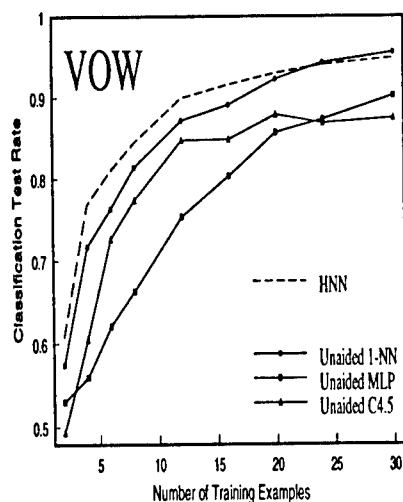


Figure 4: Test rate vs. number of training samples on the vowel data set.

the 1998 International Joint Conference on Neural Networks.

Bollacker, K. D., & Ghosh, J. (1998b). A supra-classifier architecture for scalable knowledge reuse. University of Texas Technical Report UT-CVIS-TR-98-001.

Caruana, R. (1995). Learning many related tasks at the same time with backpropagation. In *Advances in Neural Information Processing Systems 7*, pp. 657–664.

Friedman, J. H. (1994). An overview of predictive learning and function approximation. In Cherkassky, V., Friedman, J. H., & Wechsler, H. (Eds.), *From Statistics to Neural Networks, Proc. NATO/ASI Workshop*, pp. 1–61. Springer Verlag.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, CA.

Ghosh, J., & Tumer, K. (1994). Structural adaptation and generalization in supervised feedforward networks. *Journal of Artificial Neural Networks*, 1(4), 431–458.

Heckerman, D. (1997). Bayesian networks for data mining. *Data Mining and Knowledge Discovery*, 1(1), 79–120.

Mackay, D. J. C. (1995). Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3), 469–505.

Mahoney, J. J., & Mooney, R. J. (1993). Combining connectionist and symbolic learning to refine certainty factor rule bases. *Connection Science*, 5, 339–364. Nos. 3 and 4.

Mitchell, T. M. (1997). *Machine Learning*. McGraw hill, New York.

Pratt, L. Y. (1994). Experiments on the transfer of knowledge between neural networks. In Hanson, S., Drastal, G., & Rivest, R. (Eds.), *Computational Learning Theory and Natural Learning Systems, Constraints and Prospects*, chap. 19, pp. 523–560. MIT Press.

Thrun, S., & O'Sullivan, J. (1996). Discovering structure in multiple learning tasks: The TC algorithm. In *The 13th International Conference on Machine Learning*.

Towell, G., & Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2), 119–165.

Learning sorting and decision trees with POMDPs

Blai Bonet

Departamento de Computación
 Universidad Simón Bolívar
 Aptdo. 89000, Caracas 1080-A
 Venezuela
 bonet@usb.ve

Héctor Geffner

Departamento de Computación
 Universidad Simón Bolívar
 Aptdo. 89000, Caracas 1080-A
 Venezuela
 hector@usb.ve

Abstract

POMDPs are general models of sequential decisions in which both actions and observations can be probabilistic. Many problems of interest can be formulated as POMDPs, yet the use of POMDPs has been limited by the lack of effective algorithms. Recently this has started to change and a number of problems such as robot navigation and planning are beginning to be formulated and solved as POMDPs. The advantage of the POMDP approach is its clean semantics and its ability to produce principled solutions that integrate physical and information gathering actions. In this paper we pursue this approach in the context of two learning tasks: learning to sort a vector of numbers and learning decision trees from data. Both problems are formulated as POMDPs and solved by a general POMDP algorithm. The main lessons and results are that 1) the use of suitable heuristics and representations allows for the solution of sorting and classification POMDPs of non-trivial sizes, 2) the quality of the resulting solutions are competitive with the best algorithms, and 3) problematic aspects in decision tree learning such as test and misclassification costs, noisy tests, and missing values are naturally accommodated.

1 INTRODUCTION

POMDPs are general models of sequential decisions in which both actions and observations can be probabilistic (Sondik 1971; Cassandra, Kaelbling, & Littman 1994). Many problems of interest can be formulated

as POMDPs yet the use of POMDPs has been limited by the lack of effective algorithms (Cassandra, Kaelbling, & Littman 1995). Recently this has started to change and a number of problems such as robot navigation and planning are beginning to be formulated and solved as POMDPs (Cassandra, Kaelbling, & Kurien 1996; Geffner & Bonet 1998a). The advantage of the POMDP approach is its clean semantics and its ability to produce principled solutions that integrate physical and information gathering actions. In this paper we pursue this approach in the context of two learning tasks: learning to sort a vector of numbers and learning decision trees from data. Both problems are formulated as POMDPs and solved by a general POMDP algorithm (Geffner & Bonet 1998b) based on the ideas of Real Time Dynamic Programming (Barto, Bradtke, & Singh 1995).

The choice of the two tasks requires an explanation. Both are sequential decision problems that can be naturally seen as POMDPs. Yet the difficulties and insights that result from modeling and solving each problem as a POMDP are different. Sorting involves finding a sequence of comparisons and swaps that would sort *any* vector of size n . This is a challenging *planning* problem and we are not aware of any contingent planner that can model and solve problems of this type. Modeling and solving the problem from the perspective of POMDPs is challenging too. For $n = 10$, the number of possible states in the problem is greater than 10^6 . Until recently POMDPs with more than 20 states could not be reasonably solved, especially when they involved information-gathering actions. Here we provide solutions for POMDPs of size $n = 10$ that involve more than a million states. Moreover the solutions are good: on average they involve half the number of comparisons and swaps as Quicksort, one of the best sorting algorithms (Aho, Hopcroft, & Ullman 1983). The solution method relies on good heuristic func-

tions, compact representations of beliefs, and suitable decompositions.

The sorting problem is difficult and we use it not to learn about sorting but to learn about POMDPs. The focus on decision tree induction is different as we expect that the POMDP approach may contribute to a better understanding of decision tree induction on aspects such as noisy data and tests, missing values, and tests and misclassification costs. All these aspects fit into the POMDP formulation of decision tree learning in a natural way. We evaluate this formulation over a number of datasets from (Murphy & Aha 1998). Our goal is to show that the POMDP approach may be competitive with the standard approaches and potentially more general. Indeed POMDPs provide a unifying framework for modeling and solving not only sorting and induction, but other AI tasks as well such as robot navigation, planning, control, diagnosis, etc. (Cassandra, Kaelbling, & Littman 1994; Geffner & Bonet 1998a). On the other hand, the POMDPs algorithms we use do not scale up yet to learning problems over very large datasets.

The rest of the paper is organized as follows. First we review MDPs, POMDPs, and the POMDP algorithm (Sections 2 and 3). Then we formulate the problems of sorting and decision tree induction as POMDPs, and report empirical results (Sections 4 and 5). Finally we summarize the main lessons and ideas (Section 6).

2 BACKGROUND

POMDPs are a generalization of a model of sequential decision making formulated by Richard Bellman in the 50's called *Markov Decision Processes* or MDPs, in which the state of the environment is assumed known (Bellman 1957). MDPs provide the basis for understanding POMDPs so we turn to them first.¹

2.1 MDPs

The type of MDPs that we consider is a generalization of the standard search model used in AI in which actions can have *probabilistic* effects. Goal MDPs, as we call them, are characterized by:

1. a state space S

2. actions $A(s) \subseteq A$ applicable in each state s
3. positive costs $c(a, s)$ of performing action a in s
4. transition probabilities $P_a(s'|s)$ of ending up in state s' after doing action $a \in A(s)$ in state s
5. goal states $G \subseteq S$

Since the effect of actions is assumed to be *observable* but not *predictable*, the solution of an MDP is not an action sequence but a function that maps states s into actions $a \in A(s)$. Such a function is called a *policy*, and its effect is to assign a probability to each state trajectory. We assume that goal states are *absorbing* in the sense that actions in those states have no effects and zero costs. As a result, state trajectories that contain goal states have finite costs, while others have infinite costs. The *expected cost* of a policy from an initial state is the weighted average of the costs of all the state trajectories starting in that state times their probability. A policy is *optimal* when its expected cost from any state is minimal. General conditions for the existence of such policies can be found in (Puterman 1994; Bertsekas & Tsitsiklis 1996).

3 POMDPs

POMDPs generalize MDPs allowing the state to be *partially* observable (Sondik 1971; Cassandra, Kaelbling, & Littman 1994; Russell & Norvig 1994). The solution of a POMDP is no longer a mapping from states into actions, but a mapping from *belief states* into actions, where belief states are probability distributions over the states. A POMDP agent or controller starts with a prior belief state that adjusts as a result of the actions it performs and the observations it gathers. It is assumed that the agent has a model of both the actions and the sensors. Formally, a *goal* POMDP is defined in terms of:

1. **states** $s \in S$
2. **actions** $A(s) \subseteq A$ applicable in each state s
3. **positive costs** $c(a, s)$ of performing action a in s
4. **transition probabilities** $P_a(s'|s)$ of ending up in state s' after doing action $a \in A(s)$ in state s
5. **initial belief state** b_0
6. **final belief states** b_F
7. **observations** o in state s after action a with probabilities $P_a(o|s)$

The first four components define an MDP that is extended with prior and final beliefs, and a sensor model.

¹For some recent books on MDPs, see (Puterman 1994; Bertsekas & Tsitsiklis 1996); for an AI perspective, see (Boutillier, Dean, & Hanks 1995; Barto, Bradtko, & Singh 1995).

POMDPs can be formulated as *information* or *belief* MDPs in which *states* are replaced by *belief states* (Sondik 1971; Cassandra, Kaelbling, & Littman 1994). The task is to find a mapping π from belief states to actions that will take us from the initial belief state b_0 to a final belief state b_F at a minimum expected cost. The way actions and observations affect the *belief* state is given by the equations (Cassandra, Kaelbling, & Littman 1994):

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s') \quad (1)$$

$$b_a(o) = \sum_{s \in S} P_a(o|s)b_a(s) \quad (2)$$

$$b_a^o(s) = P_a(o|s)b_a(s)/b_a(o) \text{ if } b_a(o) \neq 0 \quad (3)$$

where b_a is the belief state that results after doing action a in b , $b_a(o)$ is the probability of observing o after doing a in b , and b_a^o is the belief state that results after doing action a in b and then observing o . The cost $c(a, b)$ of an action a in b is the weighted average $\sum_{s \in S} c(a, s)b(s)$. The exception are the final belief states b_F that are assumed to be absorbing; i.e., $c(a, b_F)$ is defined as 0, and b_a and b_a^o are defined as b , when b is a final belief state. Finally, the set of actions $A(b)$ applicable in b excludes the actions a that are not applicable in states s with $b(s) > 0$.

Solving belief MDPs is difficult and until recently only very small problems could be solved reasonably well especially when they involved information-gathering actions. This has started to change (Cassandra, Kaelbling, & Littman 1995) and here we use a POMDP algorithm introduced in (Geffner & Bonet 1998b) that is based on the ideas of Real Time Dynamic Programming (Barto, Bradtke, & Singh 1995).

RTDP-BEL is a *hill-climbing* algorithm that from any state b searches for the goal states b_F by performing actions a that lead to new states b_a^o with probability $b_a(o)$ (Figure 1). Estimates $V(b)$ of the expected costs to reach b_F guide the search. The main difference with standard hill-climbing is that these estimates are updated dynamically. Initially $V(b)$ is set to $h(b)$, where h is a suitable heuristic function, and every time the state b is visited $V(b)$ is updated to make it consistent with the values $V(b')$ of its possible successor states b' (Korf 1990). In the implementation, the estimates $V(b)$ are stored in a hash table that initially contains an estimate for $V(b_0)$ -only. Then when the value $V(b')$ of a state b' that is not in the table is needed, a new entry with $V(b')$ set to $h(b')$ is created. Usually belief states need to be discretized (Geffner & Bonet 1998b)

1. **Evaluate** each action a applicable in b as

$$Q(a, b) = c(a, b) + \sum_{o \in O} b_a(o)V(b_a^o)$$

initializing $V(b_a^o)$ to $h(b_a^o)$ when b_a^o not in table

2. **Apply** action a that minimizes $Q(a, b)$ breaking ties randomly
3. **Update** $V(b)$ to $Q(a, b)$
4. **Observe** o
5. **Compute** b_a^o using Equations 1–3
6. **Exit** if b_a^o is a final belief state, else set b to b_a^o and go to 1

Figure 1: RTDP-BEL

but this is not needed in the tasks considered in this paper.

RTDP-BEL combines search and simulation, and in every trial selects a random initial state s with probability $b_0(s)$ on which the effects of the actions applied by RTDP-BEL (Step 2) are simulated. More precisely, when action a is chosen, the current state s in the simulation changes to s' with probability $P_a(s'|s)$ and then produces an observation o with probability $P_a(o|s')$. The complete RTDP-BEL algorithm is shown in Fig. 1.

4 SORTING

The sorting problem involves arranging a vector of numbers in increasing order. We simplify the problem slightly assuming that no two numbers in the vector are equal. There are two types of actions available: *swap*(i, j) that exchanges the elements in positions i and j , and *cmp*(i, j) that tests whether the element in position i is smaller than the element in position j . One of the best algorithms for sorting is Quicksort, which takes in the order of $n \log(n)$ operations on average, where n is the size of the problem (the number of elements to be sorted).

4.1 FORMULATION

We formulate the problem as a goal POMDP in which we have to go from an initial belief state to a final belief state by means of a number of tests and swaps. The state s reflects the way in which the elements in the input vector may be ordered; for example, the state $s = [3, 1, 2]$ for $n = 3$ says that the first element in the input vector is the third smallest element, the second element is the smallest element of all, and the third el-

ement is the second smallest element. More generally, a state s will be a vector of size n such that $s[i] = j$, for $1 \leq i, j \leq n$ and $s[i] \neq s[j]$ for $i \neq j$. The meaning of $s[i] = j$ is that the i -th element in the input vector is the j -th smallest element.

Given an input vector, there is a single state that is the *true* state associated with the input vector and the swaps performed. The actions $cmp(i, j)$ yield information about such state and the actions $swap(i, j)$ mutate it. The resulting ‘sorting’ POMDP for a particular problem size n consists of:

1. **states** given by the vectors s of size n such that $s[i] = j$ for $0 \leq i, j \leq n$ and $s[i] \neq s[j]$ if $i \neq j$
2. **actions** $swap(i, j)$ and $cmp(i, j)$ for $0 \leq i < j \leq n$
3. **transition probabilities** $P_a(s'|s) = 1$ for $a = cmp(i, j)$ and $s' = s$, and $a = swap(i, j)$ and s' such that $s'[j] = s[i]$, $s'[i] = s[j]$, and $s'[k] = s[k]$ for $k \neq i, k \neq j$. Otherwise $P_a(s'|s) = 0$
4. **action costs** $c(a, s) = 1$ for all a and s
5. **initial belief state** b_0 uniform over all states
6. **final belief state** b_F for which $b_F(G) = 1$, where $s = G$ is the *sorted state* for which $s[i] = i$ for $i = 1, \dots, n$
7. **observations** $o_1 = (i < j)$ or $o_2 = (j < i)$ from the actions $a = test(i, j)$ with probabilities $P_a(o_1|s)$ equal to 1 (0) when $s[i] < s[j]$ ($s[i] > s[j]$), and complementary probabilities for $P_a(o_2|s)$.

4.2 IMPLEMENTATION

Finding a policy to take us from b_0 to b_F at a nearly optimal expected cost is difficult, and for the RTDP-BEL algorithm to solve this problem for even small values of n , suitable belief representations and heuristic functions are needed.

4.2.1 Representation of Beliefs

The beliefs $b(s)$ encode the probability that state s represents the way the elements in the input are ordered. For a sorting problem of size n , the size of the state space is $n!$. For $n = 10$, this means 10^6 states. Such large state spaces introduce problems of memory and time in RTDP-BEL and other POMDP algorithms. *Memory* is a potential problem as in the worst case the size of the hash table grows with the size of the

belief space which is in the order of $2^{n!}$. This problem, however, can be ameliorated by the use of good heuristic functions as discussed below.

The *time* complexity is more troublesome. The RTDP-BEL loop involves the computation of the belief states b_a and b_a^o from the original belief state b as dictated by Equations 1-3. In the worst case the time for these computations grows with $|S|^2$ and $|S||O|$ respectively. If belief states had few non-zero entries, a suitable sparse representation could be used, but this is not true in sorting where the initial belief state is uniform.

The representation that we use exploits features of the sorting problem that we expect would arise in other tasks as well.² First of all, since the prior is uniform and the ‘sensors’ (i.e., tests) are noiseless, belief states b can be represented by *sets* of states $S_b = \{s | b(s) > 0\}$. Indeed, from Bayes’ rule it follows that $b(s) = 1/|S_b|$ if $s \in S_b$ and $b(s) = 0$ otherwise. Furthermore, in sorting such sets can be conveniently encoded by collection of ‘links’ of the form $i \rightarrow j$ for $0 \leq i, j \leq n$, where each link $i \rightarrow j$ is a constraint that excludes all states s for which $s[i] \not< s[j]$. The initial belief state b_0 is represented by an empty set of such links, while the representation of b_a^o is obtained from the representation of b_a by adding the link $i \rightarrow j$ if $o = (i < j)$, and $j \rightarrow i$ if $o = (j < i)$. The representation of b_a and b are equal for $a = cmp(i, j)$ and the first is obtained from the second by exchanging the occurrences of i and j when $a = swap(i, j)$. Our implementation extends this idea with a simple mechanism that removes redundant links after any observation (a link is redundant when it can be inferred by transitivity). The result of this representation is that we reduce the complexity of updating beliefs b into b_a^o from $|S|^2$ to $|O|$ which is significantly smaller.

4.2.2 Updating the values of belief states

The structures used to represent belief states need to be converted into numbers for computing the values

$$Q(a, b) := c(a, b) + \sum_{o \in O} V(b_a^o) b_a(o)$$

This expression involves a probability $b_a(o)$ that has to be obtained from the representation of b_a . One way to compute $b_a(o)$ is by computing the proportion of states s in b_a that satisfy o (s satisfies $(i < j)$ if

²In particular we expect similar ideas to apply to the problem of handling continuous attributes in decision tree learning, but we don’t deal with such problems here.

$s[i] < s[j]$). This operation, however, is very costly as it grows linearly with $|S|$. For this reason we pursue a different approach **approximating** $b_a(o)$ for $o = (i < j)$ as:

$$b_a(o) = \begin{cases} 1 & \text{if } i \rightarrow j \text{ in } b_a \\ 0 & \text{if } j \rightarrow i \text{ in } b_a \\ 1/2 & \text{otherwise} \end{cases} \quad (4)$$

where $i \rightarrow j$ is in b_a when the link forms part of the representation of b_a or can be derived from such links by transitivity. The *approximation* here is that probabilities that are neither 0 nor 1 are mapped into 1/2. This amounts to assuming that a test $cmp(i, j)$ whose outcome is not predictable can go either way with equal probability. This assumption is not true in general but speeds up the computation and does not appear to do harm, as it is approximately correct for the tests that are optimal. We'll discuss later a similar approximation in the context of decision tree learning.

4.2.3 Heuristic Functions

The representation of beliefs reduces the complexity of updating beliefs b into b_a^o , while the approximation eliminates the cost of computing the probability $b_a(o)$. Both optimizations together speed up considerably the inner loop of the RTDP-BEL algorithm that selects and applies actions. To speed up the solution of problems we need also to consider and apply as few actions as possible. We do this by means of an heuristic function $h(b)$ that provides an estimate of the minimal expected number of actions needed to go from b to the final belief state b_F . We consider the combination of two heuristics:

1. the *longest chain heuristic* $h_l(b)$ is based on the longest sequence of links $i_1 < i_2 < i_3 < \dots < i_m$ that appear explicitly in the representation of b , with $h_l(b)$ defined as $n - m$
2. the *number of misplaced elements heuristic* $h_m(b)$ applies to *definite* belief states only; i.e., those b 's such that $b(s) = 1$ for some state s . In such a case $h_m(b)$ is defined as the number of positions $i = 1, \dots, n$, for which $s[i] \neq i$

These heuristics are not admissible in the sense that they may overestimate the minimum expected cost to the goal, and as a result may prevent the estimates $V(b)$ to approach the optimal values.³ Yet the admissible heuristics we have tried were not as informative,

³See (Barto, Bradtke, & Singh 1995) for the relation between admissibility and optimality in RTDP algorithms.

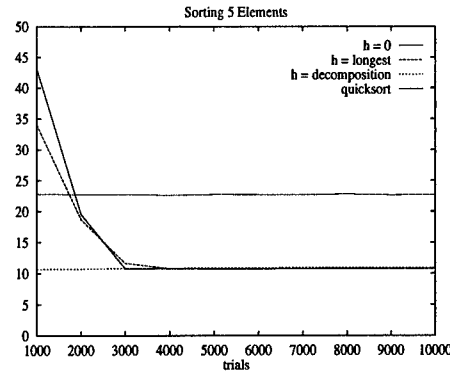


Figure 2: Average number of actions vs Number of Trials for sorting problems of sizes $n = 5$ and $n = 10$. Top line is the curve for Quicksort.

led the algorithm to visit too many belief states, and in general resulted in memory problems.

A final point about the implementation is that we impose the precondition that the ordering between the elements at positions i and j be *known* before considering a swap between them. This is done by making an action $swap(i, j)$ applicable in b only when a link $i \rightarrow j$ or $j \rightarrow i$ is in the representation of b . This condition tends to reduce the branching factor of the problem which is still large as it grows linearly with n .

4.3 EVALUATION

We tried the above implementation of the RTDP-BEL algorithm on sorting problems of two sizes. Figure 2 shows the performance of the sorting policies computed by RTDP-BEL for problems of size $n = 5$ and compares them with the ones obtained by Quicksort. The y -axis measures the average number of actions performed and the x -axis the number of trials. For $n = 5$, there are $5! = 120$ states, 20 actions, and 40 observations. The curves for RTDP-BEL correspond to the heuristic $h = 0$, $h = h_l$ and the decomposition method to be explained below. The point at trial i for $i = 1000, 2000, 3000, \dots, 10000$, indicates the average cost to reach the goal over 1000 simulations using the greedy policy determined by the estimates in the table at trial i . RTDP-BEL shows improvement with the heuristics $h = 0$ and h_l but no improvement with the decomposition method. In all cases they arrive to an expected cost that is slightly below 11 which is half the expected cost incurred by Quicksort (which is the top line in the figure). A run of 10000 trials with $h = 0$ takes in the order of 1.36 minutes and leaves 4230 entries in the hash table. The heuristic h_l and the decomposition method are slightly faster.

For larger sizes, neither of the two heuristics $h = 0$ nor $h = h_l$ scale up, and only the decomposition method works. We tried this method for $n = 10$ that generates a POMDP with several million states, 45 actions and 90 observations. The resulting curve is flat with a cost of 37. The average curve for Quicksort is also flat with an average cost of 64. The idea of the decomposition method is the following: the sorting problem is divided into two subproblems by introducing the *definite* belief states b'_F as subgoals, where the b'_F 's are such that $b'_F(s) = 1$ for some s . We deal with the problem of going from b_0 to some b'_F , and from b'_F to b_F separately. That is, each subproblem has its own heuristic function and its own hash table. The second subproblem is triggered after a belief b'_F is obtained with b'_F as the initial belief state. For the first subproblem, the heuristic h_l is used, while for the second subproblem, h_m is used.

For both $n = 5$ and $n = 10$ the resulting curves for the decomposition method are practically flat. This means that the resulting algorithm starts off well but then does not improve. As mentioned above this is the result of the non-admissibility of the heuristics h_l and h_m for each of the two subproblems. We actually ran the same algorithm for both values of n eliminating the *update* step in RTDP-BEL. The resulting algorithm is a purely *greedy* algorithm and produced the same results while consuming constant memory (the table with the estimates is not needed). However even this simplification is not good for very large values of n as the branching factor (the number of actions) grows linearly with n . For such problems other optimizations are needed. An alternative that we have considered is the use of 'indexicals' to control the actions that can be considered at any given point. The indexicals in this problem can be just a pair of vector subscripts so that only comparisons and swaps of elements with those subscripts can be considered, in addition to the operation of incrementing and decrementing those indices. Schemes such as these reduce the branching factor of the problem but push the solutions deeper in search space. Whether and when such tradeoff speeds up computation remains an open question.

4.4 SUMMARY

Sorting is a challenging problem that can be effectively modeled and solved as a POMDP provided suitable heuristics, representations and decompositions are used. In this way we have solved a POMDP that involves millions of states and have obtained solutions that compare favorably with Quicksort in terms of the

number of steps. The obvious weakness of the resulting sorting policy is that it applies to a particular problem size. An interesting challenge is the extraction of a concise and generalized representation of the policy that could be applied to problems of any size.

5 DECISION TREES

Decision trees are classifiers that map instances into classes by sequentially testing the value of a finite set of attributes (Mitchell 1997). The standard way to learn decision trees from data is by a top-down greedy strategy in which the attribute that is most informative for classification according to the data is used to split the data first, and for each possible outcome, the attribute that is most informative according to the remaining data is used second and so on, until either there are no more data or no more uncertainty regarding the classification (Breiman *et al.* 1984; Quinlan 1993). The generalization power of decision tree algorithms is measured by the classification error over part of the data that is left aside for testing. Decision tree learning algorithms have been applied to a number of domains (Murthy 1998) and a number of variations and extensions have been considered (Dietrich 1997).

5.1 FORMULATION

The problem of learning decision trees can be seen as a sequential decision problem that involves two types of actions: *report*(i) by which the current instance s is classified in class c_i , and *test*(j) by which the attribute t_j of s is observed. The goal is to have the instance s classified, and this can be achieved by any of the actions *report*(i), $i = 1, \dots, n$ where n is the number of classes. The expected cost associated with such actions depends on the *true* class of s . The actions *test*(j) provide information about s . The 'classification' POMDP consists thus of:

1. **states** s that are the instances in the training set supplemented by a separate goal state G
2. **actions** *report*(i) for each of the classes c_i , and *test*(j), for each of the attributes t_j
3. **transition probabilities** $P_a(s'|s) = 1$ for $a = \text{test}(j)$ and $s' = s$, and $a = \text{report}(i)$ and $s' = G$. Otherwise $P_a(s'|s) = 0$
4. **action costs** $c(\text{report}(i), s) = C_i$ for $\text{class}(s) = c_j$ and $c(\text{test}(j), s) = C_j$ for all s

5. **initial belief state** b_0 uniform over the non-goal states and zero over the goal state
6. **final belief state** b_F for which $b^F(G) = 1$
7. **observations** o after action $a = \text{test}(j)$ with probabilities $P_a(o|s) = 1$ if $o = v_j(s)$ and 0 otherwise, where $v_j(s)$ stands for the value of s over the attribute t_j

The POMDP formulation suggests generalizations of the standard decision tree learning setting such as different test and misclassification costs C_j and C_{ij} , noisy tests with $P_a(o|s) \in [0, 1]$, etc. By default we assume here that the cost of tests and correct classifications is 1, while the cost C_{ij} of misclassifications for $i \neq j$, is some constant $C > 1$.

5.2 IMPLEMENTATION

We represent belief states as sets of states (training set instances), taking advantage of the the uniform prior over the instances and the noiseless ‘sensors’. With this representation, the complexity of a single RTDP-BEL cycle reduces from $|S|^2$ to $|S|$. The value $b_a(o)$ for $a = \text{test}(j)$ in Equation 2 is obtained as the proportion of states s in b for which $v_j(s) = o$, a proportion that is computed as $|b_a^o|/|b|$.

We use the *non-informative* heuristic $h = 0$. Heuristics based on measures such as information gain (Quinlan 1990) could be used as well but they only make a difference in the first trials of RTDP-BEL as they are not calibrated with the expected classification costs. It may be possible to calibrate such heuristics to accelerate convergence but we don’t know how to do that yet.

5.3 EVALUATION

Table 1 compares RTDP-BEL with two standard decision tree learning algorithms, ID3 and C4.5 (Quinlan 1990; 1993) over some small datasets obtained from the UCI Repository (Murphy & Aha 1998) for two different misclassification costs C .⁴ For each dataset,

⁴The figures for ID3 and C4.5 were taken from (Friedman, Kohavi, & Yun 1996). The column named ‘Test’ in the table indicates how the generalization performance of the algorithms was measured. The Monk- n datasets come with separate training and test data; on the other two problems the test-data was generated by 5-fold cross validation: the data were partitioned into five segments, and five runs were performed by leaving one different segment as test data. The results are the averages over these five runs.

we constructed the corresponding POMDP and ran the RTDP-BEL algorithm with the non-informative heuristics $h = 0$ for 10000 trials. The curve in Figure 3 shows the average classification accuracy as a function of the number of trials in the Monk-1 and Monk-2 datasets. A run of 10000 trials over the Monk datasets takes a few minutes on average and leaves a few thousand entries in the hash table. For the larger Votes dataset, the run takes 24 minutes on average and leaves around 16000 entries in the hash table. During testing, whenever a new belief state b_a^o was generated that was not in the hash table, b_a^o was approximated to b . This means that unexpected values in the test set are regarded as ‘missing’ values. This is not too different from the approach taken in decision tree learning when test instances get to a node with no compatible branches, and are classified by the distribution of instances in that node.

5.3.1 Missing Values

In the presence of missing values in the training set, the sum of the beliefs $b_a(o)$ over the *real* observations o may fail to add up to 1 due to the mass $b_a(m) \neq 0$ over the *missing* values. In such cases, the beliefs $b_a(o)$ are normalized by dividing them by the sum $\sum_i b_a(o_i)$ taken over the *real* observations o_i . This amounts to assuming that having ‘observed’ a missing value m is like having observed a real observation o_i with probability $b_a(o_i)$. This implies that $b_a^m = b_a$, in agreement with the interpretation of missing values as *missing observations*. The dataset *Votes* in Table 1 has missing values.

5.3.2 Misclassification Costs and Overfitting

As expected, misclassification costs have an influence on the level of overfitting in noisy datasets. Very high misclassification costs induce the algorithm to fit the training data as much as possible, which in those cases may increment the error rate on the test set. This can be seen in the last row in Table 1, where the error rate in the Votes data set goes up by almost 10 points when the misclassification costs are increased from $C = 25$ to $C = 10000$. In general these costs do not have to be all equal and can be tuned to produce a minimal error rate by leaving aside part of the training data for that purpose. In other problems (e.g., medicine), these costs can be chosen to approximate the real misclassification costs.

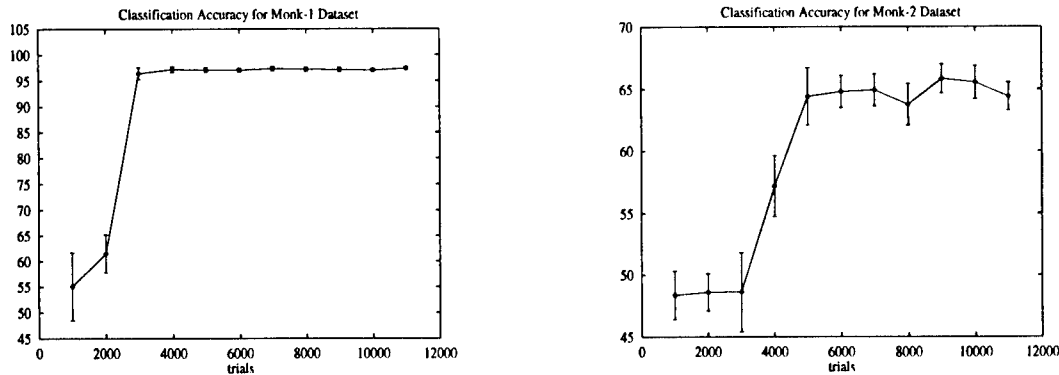


Figure 3: Classification Accuracy vs. Trials for Monk-1 and Monk-2

Table 1: Accuracy after 10000 trials compared with ID3 and C4.5

Dataset	Feat.	Miss	Train	Test	ID3	C4.5	RTDP	
							$C = 25$	$C = 10000$
monk-1	6	no	124	432	81.25 ± 1.89	75.70 ± 2.07	97.39 ± 0.29	97.39 ± 0.35
monk-2	6	no	169	432	69.91 ± 2.21	65.00 ± 2.30	64.42 ± 1.13	64.40 ± 0.81
monk-3	6	no	122	432	90.28 ± 1.43	97.20 ± 0.80	95.16 ± 0.49	94.33 ± 0.78
hayes-roth	4	no	160	CV-5	68.75 ± 8.33	74.38 ± 4.24	77.70 ± 4.65	72.04 ± 5.44
votes	16	yes	435	CV-5	93.10 ± 2.73	95.63 ± 0.43	94.42 ± 1.88	83.12 ± 6.75

5.3.3 Approximations

In another set of experiments we introduced an approximation in the evaluation of the probability $b_a(o)$, which in this case stands for the probability of observing a value v_j after testing an attribute t_j in a given context. The exact value of $b_a(o)$ is given by the number of instances in b whose attribute t_j has value v_j over the total number of instances in b . Following a similar approximation used in sorting, we approximated $b_a(o)$ uniformly as $1/n$, where n is the number of values that attribute t_j takes in the training set. As before the intuition was that the best action would be the most informative and would tend to split the data in that way. The results confirmed this intuition and matched up almost exactly the ones reported in Table 1. The CPU times were reduced three times on average. Yet even with this approximation, larger datasets could not be handled as memory tends to explode. The main problem is the lack of an informative heuristic that can guide the search, while leaving a large fraction of the (belief) state space unvisited. Heuristics such as ‘information gain’ (Quinlan 1990) are informative but are not calibrated with the expected costs.⁵ As a result, they produce a focused

search for the goal in the first few trials, but then become useless as some of the heuristic values are replaced (updated) by cost estimates. It seems that it should be possible to speed up the convergence of RTDP algorithms by the use of uncalibrated heuristics, but how to do that appears to be an open question.

5.4 SUMMARY

We have shown that decision tree induction can be modeled and solved as a POMDP problem and that solutions, while more expensive to compute, may compete in quality with the standard approaches. POMDPs may provide a fresh perspective on the problem of inferring decision trees from data as aspects such as noisy tests and data, tests and misclassification costs, and missing values, fit into the POMDP approach in a natural way. The POMDP algorithm used, however, does not scale up yet to large datasets involving many attributes, nor does it apply to datasets involving continuous attributes.

6 CONCLUSIONS

We aimed to show two things. One is that POMDPs can be used to solve complex problems of sequential decision by the use of suitable heuristics, representa-

⁵That is, information gain is not a good estimate of the expected costs.

tions, and decompositions. The second is that POMDPs provide a novel perspective on the problem of inferring decision trees from data that may be worth exploring in further depth. We have been able to solve very large POMDPs with million of states and obtain solutions that compete in quality with those produced by some of the best algorithms (Quicksort, C4.5). We expect that some of the lessons learned will be applicable to other problems such as the problem of handling continuous attributes in decision tree learning that appears to have many aspects in common with sorting. We also think that the POMDP methods used in this paper can be refined so that larger datasets could be handled. A number of interesting questions that may be relevant for the application of POMDP methods to other problems remain open; e.g., how can sorting policies be generalized to arbitrary array sizes, whether misclassification costs can be used effectively to deal with the problem of overfitting, how uncalibrated heuristics can be used to speed up converge of RTDP algorithms, etc.

Acknowledgments

This work was supported in part by a grant from Conicit, S1-96001365.

References

- Aho, A.; Hopcroft, J.; and Ullman, J. 1983. *Data Structures and Algorithms*. Addison-Wesley.
- Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81-138.
- Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.
- Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Athena Scientific.
- Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In *Proceedings of EWSP-95*.
- Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and Regression Trees*. Wadsworth International Group.
- Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete bayesian model for mobile robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robot and Systems*.
- Cassandra, A.; Kaelbling, L.; and Littman, M. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings AAAI94*, 1023-1028.
- Cassandra, A.; Kaelbling, L.; and Littman, M. 1995. Learning policies for partially observable environments: Scaling up. In *Proc. of the 12th Int. Conf. on Machine Learning*.
- Diettrich, T. 1997. Machine learning research. *Artificial Intelligence Magazine* 18(4):97-136.
- Friedman, J.; Kohavi, R.; and Yun, Y. 1996. Lazy decision trees. In *Proceedings AAAI-96*, 717-724. MIT Press.
- Geffner, H., and Bonet, B. 1998a. High-level planning and control with incomplete information using POMDP's. Available at <http://www ldc.usb.ve/~hector>.
- Geffner, H., and Bonet, B. 1998b. Solving large POMDPs using real time dynamic programming. Available at <http://www ldc.usb.ve/~hector>.
- Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189-211.
- Mitchell, T. 1997. *Machine Learning*. McGraw-Hill.
- Murphy, P. M., and Aha, D. W. 1998. UCI repository of machine learning databases. <http://www.ics.uci.edu/learn>.
- Murthy, S. 1998. Automatic construction of decision trees from data: A multidisciplinary survey. Technical report, Siemens Corporate Research.
- Puterman, M. 1994. *Markov Decision Processes - Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc.
- Quinlan, J. R. 1990. Induction of decision trees. *Machine Learning* 1(1).
- Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufman.
- Russell, S., and Norvig, P. 1994. *Artificial Intelligence: A Modern Approach*. Prentice Hall.
- Sondik, E. 1971. *The Optimal Control of Partially Observable Markov Processes*. Ph.D. Dissertation, Stanford University.

Feature Selection via Concave Minimization and Support Vector Machines

P. S. Bradley

Computer Sciences Department
University of Wisconsin
Madison, WI 53706
paulb@cs.wisc.edu

O. L. Mangasarian

Computer Sciences Department
University of Wisconsin
Madison, WI 53706
olvi@cs.wisc.edu

Abstract

Computational comparison is made between two feature selection approaches for finding a separating plane that discriminates between two point sets in an n -dimensional feature space that utilizes as few of the n features (dimensions) as possible. In the concave minimization approach [19, 5] a separating plane is generated by minimizing a weighted sum of distances of misclassified points to two parallel planes that bound the sets and which determine the separating plane midway between them. Furthermore, the number of dimensions of the space used to determine the plane is minimized. In the support vector machine approach [27, 7, 1, 10, 24, 28], in addition to minimizing the weighted sum of distances of misclassified points to the bounding planes, we also *maximize* the distance between the two bounding planes that generate the separating plane. Computational results show that feature suppression is an indirect consequence of the support vector machine approach when an appropriate norm is used. Numerical tests on 6 public data sets show that classifiers trained by the concave minimization approach and those trained by a support vector machine have comparable 10-fold cross-validation correctness. However, in all data sets tested, the classifiers obtained by the concave minimization approach selected fewer problem features than those trained by a support vector machine.

1 INTRODUCTION

The feature selection problem addressed here is that of discriminating between two finite point sets in n -dimensional feature space R^n by a separating plane that utilizes as few of the features as possible.

Classification performance is determined by the inherent class information available in the features provided. It seems logical to conclude that a large number of features would provide more discriminating ability. But, with a finite training sample, a high-dimensional feature space is almost empty [12] and many separators may perform well on the training data, but few may generalize well. Hence the importance of the feature selection problem in classification [15]. The optimization formulations in Section 2 exploit one realization of the Occam's Razor bias [3]: compute a separating plane with a small number of predictive features, discarding irrelevant or redundant features. These formulations can be considered *wrapper models* as defined in [14].

The first approach [19, 5], described in Section 2, involves the minimization of a concave function on a polyhedral set. A plane is constructed such that a weighted sum of distances of misclassified points to the plane is minimized and as few dimensions of the original feature space R^n are used. This is achieved by constructing two parallel bounding planes, in as small dimensional space as possible, that bound each of the two sets to the extent possible by placing the two sets on two opposite halfspaces determined by the two planes. The two planes are determined such that the sum of weighted distances of points in the wrong halfspace to the bounding plane is minimized. This leads to the minimization of a concave function on a polyhedral set (problems (6) and (8) below) for which a stationary point can be obtained a successive lin-

earization algorithm (Algorithm 2.1 below). The final separating plane is taken midway between the two bounding parallel planes.

The second approach, that of a support vector machine [27, 7, 1, 10, 24, 28], described in Section 3, constructs two parallel bounding planes in n -dimensional space R^n as in the first approach outlined above, but in addition attempts to push these planes as far apart as possible. The justification for this, apart from reducing the VC dimension [27] which in turn improves generalization, is that for the linearly separable case, the further apart the planes, the smaller the halfspace assigned to each of the two sets, reducing the possibility that new unseen points from the wrong set lie in that halfspace. Although improved generalization is the primary purpose of the support vector machine formulation, it turns out that the linear program (13) resulting from employing the ∞ -norm to measure the distance between the two bounding planes, leads also to a feature selection method, whereas the linear program resulting from the use of the 1-norm (12) and the quadratic program resulting from the 2-norm (14) do not lead to feature selection methods.

In Section 4 we describe our computational experiments on 6 publicly available data sets using the approaches described in Sections 2 and 3. The goal is to evaluate the generalization ability of classifiers trained by solving: the concave optimization problem (8), three versions of the support vector machine problem with different norms (12), (13), (14) as well as the robust linear program RLP (4). RLP, which underlies the proposed feature selection methods here, has no feature suppression capability built in. We measure generalization ability by 10-fold cross-validation [26]. Numerical tests on 6 public data sets show that classifiers trained by the concave minimization approach and those trained by a support vector machine have comparable 10-fold cross-validation correctness. However, in all data sets tested, the classifiers obtained by the concave minimization approach selected fewer problem features than those trained by a support vector machine. Further, computational time for the normally used quadratic programming approach for SVMs, was orders of magnitude larger than the proposed linear programming approaches.

We now describe our notation and give some background material. All vectors will be column vectors unless transposed to a row vector by a superscript T . For a vector x in R^n , $|x|$ will denote a vector in R^n of absolute values of the components of x . For a vector $x \in R^n$, x_+ denotes the vector in R^n with components

$\max\{0, x_i\}$. For a vector $x \in R^n$, x_* denotes the vector in R^n with components $(x_*)_i = 1$ if $x_i > 0$ and 0 otherwise (i.e. x_* is the result of applying the step function component-wise to x). The base of the natural logarithm will be denoted by ε , and for a vector $y \in R^m$, ε^{-y} will denote a vector in R^m with components ε^{-y_i} , $i = 1, \dots, m$. For $x \in R^n$ and $1 \leq p < \infty$:

$$\|x\|_p = \left(\sum_{j=1}^n |x_j|^p \right)^{\frac{1}{p}}, \quad \|x\|_\infty = \max_{1 \leq j \leq n} |x_j|.$$

For a general norm $\|\cdot\|$ on R^n , the *dual norm* $\|\cdot\|'$ on R^n is defined as

$$\|x\|' = \max_{\|y\|=1} x'y.$$

The 1-norm and ∞ -norm are dual norms, and so are a p -norm and a q -norm for which $1 \leq p, q \leq \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$. The notation $A \in R^{m \times n}$ will signify a real $m \times n$ matrix. For such a matrix A^T will denote the transpose of A and A_i will denote the i -th row of A . A vector of ones in a real space of arbitrary dimension will be denoted by e . A vector of zeros in a real space of arbitrary dimension will be denoted by 0. The notation $\arg \min_{x \in S} f(x)$ will denote the set of minimizers of $f(x)$ on the set S . A separating plane, with respect to two given point sets \mathcal{A} and \mathcal{B} in R^n , is a plane that attempts to separate R^n into two halfspaces such that each open halfspace contains points mostly of \mathcal{A} or \mathcal{B} .

2 FSV: FEATURE SELECTION VIA CONCAVE MINIMIZATION

In this part of the paper we describe a feature selection procedure that has been effective in medical and other applications [5, 19].

Given two point sets \mathcal{A} and \mathcal{B} in R^n represented by the matrices $A \in R^{m \times n}$ and $B \in R^{k \times n}$ respectively, we wish to discriminate between them by a separating plane:

$$P = \{x \mid x \in R^n, x^T w = \gamma\}, \quad (1)$$

with normal $w \in R^n$ and 1-norm distance to the origin of $\frac{|\gamma|}{\|w\|_\infty}$ [20]. We shall attempt to determine w and γ so that the separating plane P defines two open halfspaces $\{x \mid x \in R^n, x^T w > \gamma\}$ containing mostly points of \mathcal{A} , and $\{x \mid x \in R^n, x^T w < \gamma\}$ containing mostly

points of \mathcal{B} . Hence, upon normalization, we wish to satisfy

$$Aw \geq e\gamma + e, Bw \leq e\gamma - e. \quad (2)$$

to the extent possible. Conditions (2) can be satisfied if and only if, the convex hulls of \mathcal{A} and \mathcal{B} are disjoint. This is not the case in many real-world applications. Hence, we attempt to satisfy (2) in some “best” sense by minimizing some norm of the average violations of (2) such as

$$\min_{w, \gamma} f(w, \gamma) = \min_{w, \gamma} \frac{1}{m} \|(-Aw + e\gamma + e)_+\|_1 + \frac{1}{k} \|(Bw - e\gamma + e)_+\|_1. \quad (3)$$

Recall that for a vector x , x_+ denotes the vector with components $\max\{0, x_i\}$. Two principal reasons for choosing the 1-norm in (3) are: (1) problem (3) is then reducible to a linear program (4) with many important theoretical properties making it an effective computational tool [2], (2) the 1-norm is less sensitive to outliers such as those occurring when the underlying data distributions have pronounced tails, hence (3) has a similar effect to that of robust regression [13], [11, pp 82-87].

The formulation (3) is equivalent to the following robust linear programming formulation (RLP) proposed in [2] and effectively used to solve problems from real-world domains [21]:

$$\begin{aligned} & \text{minimize}_{w, \gamma, y, z} && \frac{e^T y}{m} + \frac{e^T z}{k} \\ & \text{subject to} && -Aw + e\gamma + e \leq y, \\ & && Bw - e\gamma + e \leq z, \\ & && y \geq 0, z \geq 0. \end{aligned} \quad (4)$$

The linear program (4) or, equivalently, the formulation (3), define a separating plane P that approximately satisfies the conditions (2) in the following sense. Each positive value of y_i determines the distance $\frac{y_i}{\|w\|}$ [20, Theorem 2.2] between a point A_i of \mathcal{A} lying on the wrong side of the bounding plane $x^T w = \gamma + 1$ for \mathcal{A} , that is A_i lying in the open half-space

$$\{x \mid x^T w < \gamma + 1\},$$

and the bounding plane $x^T w = \gamma + 1$. Similarly for \mathcal{B} and $x^T w = \gamma - 1$. Thus the objective function of

the linear program (4) minimizes the average sum of distances, weighted by $\|w\|$, of misclassified points to the bounding planes. The separating plane P (1) is midway between the two bounding planes and parallel to them.

Feature selection [19, 5] is imposed by attempting to suppress as many components of the normal vector w to the separating plane P that is consistent with obtaining an acceptable separation between the sets \mathcal{A} and \mathcal{B} . We achieve this by introducing an extra term with parameter $\lambda \in [0, 1)$ into the objective of (4) while weighting the original objective by $(1 - \lambda)$ as follows:

$$\begin{aligned} & \text{minimize}_{w, \gamma, y, z} && (1 - \lambda) \left(\frac{e^T y}{m} + \frac{e^T z}{k} \right) + \lambda e^T |w|_* \\ & \text{subject to} && -Aw + e\gamma + e \leq y, \\ & && Bw - e\gamma + e \leq z, \\ & && y \geq 0, z \geq 0. \end{aligned} \quad (5)$$

Note that the vector $|w|_* \in R^n$ has components which are equal to 1 if the corresponding components of w are nonzero and components equal to zero if the corresponding components of w are zero. Recall that e is a vector of ones and $e^T |w|_*$ is simply a count of the nonzero elements in the vector w . Problem (5) balances the error in separating the sets \mathcal{A} and \mathcal{B} , $\left(\frac{e^T y}{m} + \frac{e^T z}{k} \right)$, and the number of nonzero elements of w , $(e^T |w|_*)$. Further, if an element of w is zero, the corresponding feature is removed from the problem.

By introducing the variable v we are able to eliminate the absolute value from problem (5) which leads to the following equivalent parametric program (for $\lambda \in [0, 1)$):

$$\begin{aligned} & \text{minimize}_{w, \gamma, y, z, v} && (1 - \lambda) \left(\frac{e^T y}{m} + \frac{e^T z}{k} \right) + \lambda e^T v_* \\ & \text{subject to} && -Aw + e\gamma + e \leq y, \\ & && Bw - e\gamma + e \leq z, \\ & && y \geq 0, z \geq 0, \\ & && -v \leq w \leq v. \end{aligned} \quad (6)$$

Since v appears positively weighted in the objective and is constrained by $-v \leq w \leq v$, it effectively models the vector $|w|$. This feature selection problem will be solved for a value of $\lambda \in [0, 1)$ for which the resulting classification obtained by the separating plane (1) midway between the bounding planes $x^T w = \gamma \pm 1$,

generalizes best, estimated by a cross-validation tuning procedure. Typically this will be achieved in a feature space of reduced dimensionality, that is $e^T v_* < n$ (i.e. the number of features used is less than n).

Because of the discontinuity of the step function term $e^T v_*$, we approximate it by a concave exponential on the nonnegative real line [19]. The approximation of the step vector v_* of (6) by the concave exponential :

$$v_* \approx t(v, \alpha) = e - \varepsilon^{-\alpha v}, \alpha > 0, \quad (7)$$

leads to the smooth problem (**FSV**:Feature Selection Concave):

$$\begin{aligned} \text{minimize}_{w, \gamma, y, z, v} \quad & (1 - \lambda) \left(\frac{e^T y}{m} + \frac{e^T z}{k} \right) + \lambda e^T (e - \varepsilon^{-\alpha v}) \\ & -Aw + e\gamma + e \leq y, \\ \text{subject to} \quad & Bw - e\gamma + e \leq z, \\ & y \geq 0, z \geq 0, \\ & -v \leq w \leq v. \end{aligned} \quad (8)$$

It can be shown [4, Theorem 2.1] that for a finite value of α (appearing in the concave exponential) the smooth problem (8) generates an exact solution of the nonsmooth problem (6). We note that this problem is the minimization of a concave objective function over a polyhedral set. Even though it is difficult to find a global solution to this problem, a fast successive linear approximation (SLA) algorithm [5, Algorithm 2.1] terminates finitely (usually in 5 to 7 steps) at a stationary point which satisfies the minimum principle necessary optimality condition for problem (8) [5, Theorem 2.2] and leads to a sparse w with good generalization properties. For convenience we state the SLA algorithm below.

Algorithm 2.1

Successive Linearization Algorithm (SLA) for FSV (8). Choose $\lambda \in [0, 1]$. Start with a random $(w^0, \gamma^0, y^0, z^0, v^0)$. Having $(w^i, \gamma^i, y^i, z^i, v^i)$ determine $(w^{i+1}, \gamma^{i+1}, y^{i+1}, z^{i+1}, v^{i+1})$ by solving the linear program:

$$\begin{aligned} \text{minimize}_{w, \gamma, y, z, v} \quad & (1 - \lambda) \left(\frac{e^T y}{m} + \frac{e^T z}{k} \right) + \lambda \alpha \left(\varepsilon^{-\alpha v^i} \right)^T (v - v^i) \\ & -Aw + e\gamma + e \leq y, \\ \text{subject to} \quad & Bw - e\gamma + e \leq z, \\ & y \geq 0, z \geq 0, \\ & -v \leq w \leq v. \end{aligned} \quad (9)$$

Stop when

$$(1 - \lambda) \left(\frac{e^T (y^{i+1} - y^i)}{m} + \frac{e^T (z^{i+1} - z^i)}{k} \right) + \lambda \alpha \left(\varepsilon^{-\alpha v^i} \right)^T (v^{i+1} - v^i) = 0. \quad (10)$$

Comment: The parameter α was set to 5. The parameter λ was chosen to "maximize" generalization performance.

We have found useful solutions to (8) for the fixed value $\alpha = 5$ [5, 4]. Another approach, involving more computation, is to solve (8) for an increasing sequence of α values.

3 SVM: FEATURE SELECTION VIA SUPPORT VECTOR MACHINES

The support vector machine idea [27, 1, 10, 24, 28], although not originally intended as a feature selection tool, does in fact indirectly suppress components of the normal vector w to the separating plane P (1) when an appropriate norm is used for measuring the distance between the two parallel bounding planes for the sets being separated. The SVM approach consists of adding another term, $\frac{\|w\|'}{2}$, to the objective function of the RLP (4) in a similar manner to the appended term $e^T |w|_*$ of problem (5). Here, $\|\cdot\|'$ is the dual of some norm on R^n used to measure the distance between the two bounding planes. The justification for this term is as follows. The separating plane P (1) generated by the RLP linear program (4) lies midway between the two parallel planes $w^T x = \gamma + 1$ and $w^T x = \gamma - 1$. The distance, measured by some norm $\|\cdot\|$ on R^n , between these planes is precisely $\frac{2}{\|w\|}$ [20, Theorem 2.2]. The appended term to the objective function of the RLP (4), $\frac{\|w\|'}{2}$, is the reciprocal of this distance, thus driving the distance between these two planes up to obtain better separation. This results then in the following mathematical programming formulation for the SVM formulation:

$$\begin{aligned} \text{minimize}_{w, \gamma, y, z, v} \quad & (1 - \lambda) (e^T y + e^T z) + \frac{\lambda}{2} \|w\|' \\ & -Aw + e\gamma + e \leq y, \\ \text{subject to} \quad & Bw - e\gamma + e \leq z, \\ & y \geq 0, z \geq 0. \end{aligned} \quad (11)$$

Points $A_i \in \mathcal{A}$ and $B_i \in \mathcal{B}$ appearing in active constraints of the linear program (11) with positive dual

variables constitute the *support vectors* of the problem. These points are the *only* data points that are relevant for determining the optimal separating plane. Their number is usually small and it is proportional to the generalization error of the classifier [24].

If we use the 1-norm to measure the distance between the planes, then the dual to this norm is the ∞ -norm and accordingly $\|w\|' = \|w\|_\infty$ in (11) which leads to the following linear programming formulation:

$$\begin{aligned} & \underset{w, \gamma, y, z, \nu}{\text{minimize}} && (1 - \lambda)(e^T y + e^T z) + \frac{\lambda}{2} \nu \\ & \text{subject to} && \begin{aligned} & -Aw + e\gamma + e \leq y, \\ & Bw - e\gamma + e \leq z, \\ & -e\nu \leq w \leq e\nu, \\ & y \geq 0, z \geq 0. \end{aligned} \end{aligned} \quad (12)$$

Similarly if we use the ∞ -norm to measure the distance between the planes, then the dual to this norm is the 1-norm and accordingly $\|w\|' = \|w\|_1$ in (11) which leads to the following linear programming formulation:

$$\begin{aligned} & \underset{w, \gamma, y, z, s}{\text{minimize}} && (1 - \lambda)(e^T y + e^T z) + \frac{\lambda}{2} e^T s \\ & \text{subject to} && \begin{aligned} & -Aw + e\gamma + e \leq y, \\ & Bw - e\gamma + e \leq z, \\ & -s \leq w \leq s, \\ & y \geq 0, z \geq 0. \end{aligned} \end{aligned} \quad (13)$$

We note that the first paper on the multisurface method on pattern separation [17] also proposed and implemented, just as does the support vector machine approach, forcing the two parallel planes that bound the sets to be separated to be as far apart as possible.

Usually the support vector machine problem is formulated using the 2-norm in the objective [27, 1]. Since the 2-norm is dual to itself, it follows that the distance between the parallel planes defining the separating surface is also measured in the 2-norm when this formulation is used. In this case $\|w\|' = \|w\|_2$, and one usually appends the term $\frac{\lambda}{2} \|w\|_2^2$ to the objective of (11) resulting in the following quadratic program:

$$\begin{aligned} & \underset{w, \gamma, y, z}{\text{minimize}} && (1 - \lambda)(e^T y + e^T z) + \frac{\lambda}{2} w^T w \\ & \text{subject to} && \begin{aligned} & -Aw + e\gamma + e \leq y, \\ & Bw - e\gamma + e \leq z, \\ & y \geq 0, z \geq 0. \end{aligned} \end{aligned} \quad (14)$$

Nonlinear separating surfaces, which are linear in their parameters, can also easily be handled by the formulations (8), (12) and (13) [16]. If the data are mapped nonlinearly via $\Phi: R^n \rightarrow R^l$, a nonlinear separating

surface in R^n is easily computed as a linear separator in R^l . In practice, one usually solves (14) by way of its dual [18]. In this formulation, the data enter only as inner products which are computed in the *transformed* space via a kernel function $K(x, y) = \Phi(x) \cdot \Phi(y)$ [6, 27, 28].

We note that separation errors in (12) - (14) are weighted equally conforming to the SVM formulations in [6, 27]. In contrast, the formulations (4) and (8) measure *average* separation error. Minimizing average separation error in (4) ensures that the solution $w = 0$ occurs iff $\frac{e^T A}{m} = \frac{e^T B}{k}$, in which case it is not unique [2, Theorem 2.5].

We turn our attention now to computational testing and comparison.

4 COMPUTATIONAL RESULTS

4.1 DATA SETS

The Wisconsin Prognostic Breast Cancer Database consists of 198 instances with 35 features representing follow-up data for one breast cancer case [23].

We used 2 variants of this data set. The first data set was created where the elements of the set \mathcal{A} were 30 nuclear features plus diameter of excised tumor and number of positive lymph nodes of instances corresponding to patients in which cancer had recurred in less than 24 months (28 points). The set \mathcal{B} consisted of the same features for patients in which cancer had not recurred in less than 24 months (127 points). The second variant of the data set consisted of the same 32 features, but splits the data into \mathcal{A} and \mathcal{B} differently. Elements of \mathcal{A} corresponds to patients with a cancer recurrence in less than 60 months (41 points) and \mathcal{B} corresponds to patients which cancer had not recurred in less than 60 months (69 points).

The Johns Hopkins University Ionosphere data set consists of 34 continuous features of 351 instances [23]. Each instance represents a radar return from the ionosphere. The set \mathcal{A} consists of 225 radar returns termed "good" or showing some type of structure in the ionosphere. The set \mathcal{B} consists of 126 radar returns termed "bad"; their signals pass through the ionosphere.

The Cleveland Heart Disease data set consists of 297 instance with 13 features (see documentation [23]). Set \mathcal{A} consist of 214 instance. The set \mathcal{B} consists of 83 instances.

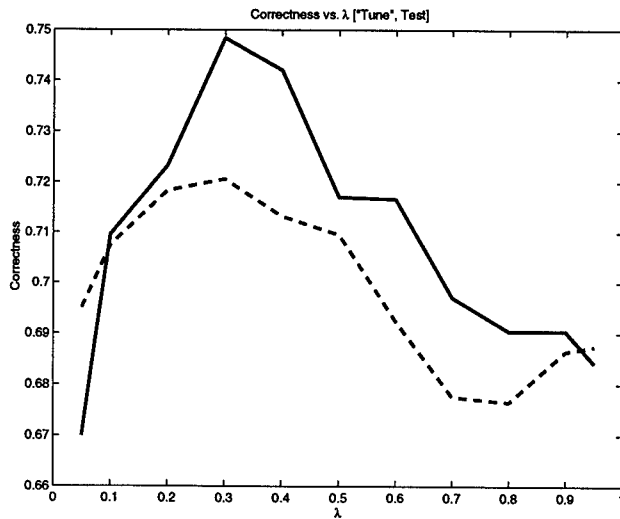


Figure 1: Tuning and testing sets correctness for a support vector machine (13) versus the sparsity-inducing parameter λ on the WPBC (24 months) data set. **Dashed** = “tuning” correctness, **Solid** = test correctness.

The Pima Indians Diabetes data set consists of 768 instances with 8 features plus a class label (see documentation [23]). The 500 instances with class label “0” were placed in \mathcal{A} , the 268 instances with class label “1” were placed in \mathcal{B} .

The BUPA Liver Disorders data set consists of 345 instances with 6 features plus a selector field used to split the data into 2 sets (see documentation [23]). Set \mathcal{A} consists of 145 instances, the set \mathcal{B} consists of 200 instances.

4.2 EXPERIMENTAL METHODOLOGY

Our goal was to evaluate the generalization ability of the classifiers obtained by solving: the concave minimization problem FSV (8), SVM 1-norm problem (13), the SVM ∞ -norm problem (12), the SVM 2-norm problem (14) and the robust linear program (RLP) (4). We estimate the generalization ability of a classifier via 10-fold cross-validation [26].

We note that the objective function parameter λ , which can induce sparsity, must be chosen carefully to maximize the generalization ability of the resulting classifier. Choosing $\lambda = 0$ will maximize the *training* correctness of the resulting classifier, but often this classifier performs poorly on data not in the training set [25]. We employ the following “tuning set” procedure for choosing λ at each fold of 10-fold cross-validation: For each λ in a candidate set Λ , we perform the following: (i) set aside 10% of the training data as

a “tuning” set, (ii) obtain a classifier for the given value of λ , (iii) determine correctness on the “tuning” set, (iv) repeat steps (i)-(iii) ten times, each time setting aside a different 10% portion of the training data. The “score” for this value of λ is the average of the 10 correctness values determined in (iii).

We fix the value of λ as that with the best “score” determined from the tuning procedure (ties are broken by choosing the smallest λ -value). This is the value used for the given fold of 10-fold cross-validation. The set Λ is a set of candidate values and for these experiments was set at: $\Lambda = \{0.05, 0.10, 0.20, \dots, 0.90, 0.95\}$. The curves in Figure 1 indicate that the value of λ that maximizes the “tuning” score (dashed curve in Figure 1) is a good estimate of the value of λ that maximizes the test set correctness (solid curve).

4.3 EXPERIMENTAL RESULTS

Table 1 summarizes the average number of original problem features selected by the classifiers trained by each of the methods.

Table 2 summarizes the results of the 10-fold cross-validation experiments on 6 real-world data sets. All “Train” and “Test” numbers presented are average correctnesses over 10-folds. The p -value is an indicator of significance difference in “Test” correctness between the classifiers obtained by solving FSV (8) and the classifiers obtained by solving the SVM 1-norm problem (13)¹. Recall that a high p -value indicates that the difference is not significant. We note that p -values were not calculated for the other pairwise comparisons because the solutions obtained by solving the SVM ∞ -norm, SVM 2-norm and the RLP did not suppress problem features (see Table 1).

4.4 DISCUSSION

The FSV (8) and the SVM 1-norm (13) problems were the only ones exhibiting feature selection (Table 1). On the 6 data sets tested, the SVM 1-norm classifiers performed slightly better on 3 data sets and FSV classifiers performed slightly better on 3 data sets. The minimum p -value is 0.1246 indicates that classifiers obtained by the FSV (8) and the SVM 1-norm (13) methods have similar generalization properties. Applying the paired t -test to 10-fold cross validation results may indicate a difference in the average test

¹Specifically, this is the p -value of a two-tailed paired t -test testing the hypothesis that the difference in “Test” correctnesses for the FSV and SVM 1-norm classifiers is zero

set correctness when one is not present [9]. Thus the results of these experiments may be more similar than indicated by the p -values.

We note that the classifiers obtained by solving the SVM ∞ -norm (12) suppressed none of the original problem features for all but the largest values of λ (near 1.0), which in general is of little use because it is often accompanied by poor set separation. Similar behavior was observed by solving the SVM 2-norm (14) problem. Note that the ∞ -norm is sensitive to outliers, as is the 2-norm *squared*.

The classifiers obtained by solving the FSV problem (8) selected fewer problem features than the any of the SVM formulations (12), (13), (14) and the RLP (4). FSV classifiers reduced the number of features used over SVM 1-norm by as much as 39.5% (WPBC 60 month), while maintaining comparable generalization performance.

On the WPBC 24 month dataset, both the FSV classifiers (8) and the SVM 1-norm classifiers (13) most often selected a nuclear area feature and number of lymph nodes removed from the patient. These features are deemed relevant to the prognosis problem.

All linear programs formulations were solved using the CPLEX package [8] called from within MATLAB [22]. The quadratic programming problem (14) was solved using MATLAB's quadratic optimization solver, which encountered difficulty on conditioning the QP constraint matrix, which may affect the interpretation of the results for this approach. See Table 3 for average solve times.

5 SUMMARY AND FUTURE WORK

Computational comparisons of classifiers obtained by solving four mathematical optimization problems are presented. The optimization formulations are either linear (4), (12) and (13), or quadratic (14), or can be solved by a finite sequence of linear programs (solving (8) via Algorithm 2.1). **Classifiers obtained by solving the FSV problem (8) and the SVM 1-norm problem (13) exhibit feature suppression and have comparable generalization performance on six publicly available real world data sets tested. The classifiers obtained by solving the FSV problem (8) suppressed more features than the corresponding SVM 1-norm classifiers (13). The quadratic SVM (14) took orders of magnitude more time than the linear-**

programming-based SVMs (12) and (13).

When the distance between the 2 parallel planes defining the separating surface in the SVM problem is chosen to be the 1-norm, the resulting SVM optimization problem has the ∞ -norm (dual norm to the 1-norm) appearing in the objective. The classifiers obtained by solving this problem (SVM ∞ -norm (12)) did not exhibit feature selection. Similar behavior was observed for classifiers obtained by solving the SVM 2-norm (14) problem. The generalization ability of these classifiers in comparison with the others presented needs to be further investigated.

Future work includes further analysis of the benefits of measuring the distance between the bounding parallel planes defining the separating plane and the resulting optimization problem utilizing the dual norm (11). A characterization of classes of data sets which lend themselves to better separation with the choice of one norm over another will allow practitioners to choose *a priori* an optimization formulation believed to be "best" suited to the separation problem at hand.

Acknowledgements

This work was supported by National Science Foundation Grants CCR-9322479, CCR-9729842 and Air Force Office of Scientific Research Grant F49620-97-1-0326 as Mathematical Programming Technical Report 98-03, February 1998.

References

- [1] K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. Department of Mathematical Sciences Math Report No. 97-100, Rensselaer Polytechnic Institute, Troy, NY 12180, 1997. <http://www.math.rpi.edu/bennek/>.
- [2] K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23-34, 1992.
- [3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377-380, 1987.
- [4] P. S. Bradley, O. L. Mangasarian, and J. B. Rosen. Parsimonious least norm approximation. Technical Report 97-03, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, March

Table 1: Average number of features selected in the classifiers. (Asterisk * indicates that the full experiment had not been carried out because of excessive time (see Table 3), hence results are averaged over folds completed.)

Data Set	FSV (8)	SVM $\ \cdot\ _1$ (13)	SVM $\ \cdot\ _\infty$ (12)	SVM $\ \cdot\ _2^2$ (14)	RLP (4)
WPBC (24 mo.)	3.9	5.4	32	32	32
WPBC (60 mo.)	2.6	4.3	32	32	32
Ionosphere	10.4	11.1	34	33*	33
Cleveland	6.4	9.3	13	13*	13
Pima Indians	5.3	6.0	8	*	8
BUPA Liver	4.5	5.8	6	6*	6

Table 2: Ten-fold cross-validation correctness results on 6 publicly available data sets. (Asterisk * indicates that the full experiment had not been carried out because of excessive time (see Table 3), hence results are averaged over folds completed.)

Data Set	FSV (8) Train Test	SVM $\ \cdot\ _1$ (13) Train Test	p-Value	SVM $\ \cdot\ _\infty$ (12) Train Test	SVM $\ \cdot\ _2^2$ (14) Train Test	RLP (4) Train Test
WPBC (24 mo.)	73.97 66.42	74.40 71.08	0.1246	73.69 71.04	82.86 75.46	85.23 67.12
WPBC (60 mo.)	70.70 67.05	71.21 66.23	0.6408	73.34 66.38	75.54 66.21	87.58 63.50
Ionosphere	90.47 84.07	88.92 86.10	0.1254	89.65 84.06	94.56* 85.75*	94.78 86.04
Cleveland	83.57 80.94	85.30 84.55	0.1819	85.82 82.52	84.70* 75.86*	86.31 83.87
Pima Indians	75.22 74.60	75.52 74.47	0.8889	76.01 74.99	* *	76.48 76.16
BUPA Liver	68.18 65.20	67.83 64.03	0.1696	68.73 64.63	60.22* 60.95*	68.98 64.34

1997. *Computational Optimization and Applications*, to appear. <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-03.ps.Z>.

mizer and CPLEX(TM) Mixed Integer Optimizer (Version 2.0), 1992.

- [5] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Feature selection via mathematical programming. *INFORMS Journal on Computing*, 1998. To appear. <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-21.ps.Z>.
- [6] C. J. C. Burges. A tutorial on support vector machines. *Data Mining and Knowledge Discovery*, 2, 1998. To appear.
- [7] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–279, 1995.
- [8] CPLEX Optimization Inc., Incline Village, Nevada. *Using the CPLEX(TM) Linear Opti-*
- [9] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 1997. To appear. <http://www.cs.orst.edu/tgd/cv/pubs.html>.
- [10] F. Girosi. An equivalence between sparse approximation and support vector machines. A.I. Memo 1606, Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts, 1997. <http://www.ai.mit.edu/people/girosi/homepage/svm.html>.
- [11] M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1995.

Table 3: Average running: Ionosphere data set.

Method	Time (Seconds)
Algorithm 2.1	30.94
$\ \cdot\ _1$ (13)	3.09
$\ \cdot\ _\infty$ (12)	1.42
$\ \cdot\ _2^2$ (14)	2956.8
RLP (4)	1.28

- [12] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, California, 1991.
- [13] P. J. Huber. *Robust Statistics*. John Wiley, New York, 1981.
- [14] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning*, San Mateo, CA, 1994. Morgan Kaufmann.
- [15] D. Koller and M. Sahami. Toward optimal feature selection. In L. Saitta, editor, *Machine Learning—Proceedings of the Thirteenth International Conference (ICML '96)—Bari, Italy July 3-6, 1996*, pages 284–292, San Francisco, CA, 1996. Morgan Kaufmann.
- [16] O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.
- [17] O. L. Mangasarian. Multi-surface method of pattern separation. *IEEE Transactions on Information Theory*, IT-14:801–807, 1968.
- [18] O. L. Mangasarian. *Nonlinear Programming*. McGraw-Hill, New York, 1969. Reprint: SIAM Classic in Applied Mathematics 10, 1994, Philadelphia.
- [19] O. L. Mangasarian. Machine learning via polyhedral concave minimization. In H. Fischer, B. Riedmueller, and S. Schaeffler, editors, *Applied Mathematics and Parallel Computing - Festschrift for Klaus Ritter*, pages 175–188. Physica-Verlag A Springer-Verlag Company, Heidelberg, 1996. <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-20.ps.Z>.
- [20] O. L. Mangasarian. Arbitrary-norm separating plane. Technical Report 97-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, May 1997. *Operations Research Letters*, submitted. <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-07.ps.Z>.
- [21] O. L. Mangasarian, W. N. Street, and W. H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, July-August 1995.
- [22] MATLAB. *User's Guide*. The MathWorks, Inc., 1992.
- [23] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Technical report, Department of Information and Computer Science, University of California, Irvine, 1992. www.ics.uci.edu/mllearn/MLRepository.html.
- [24] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *IEEE Conference on Computer Vision and Pattern Recognition, Puerto Rico, June 1997, 130-136*, 1997. <http://www.ai.mit.edu/people/girosi/home-page/svm.html>.
- [25] J. W. Shavlik and T. G. Dietterich (editors). *Readings in Machine Learning*. Morgan Kaufman, San Mateo, California, 1990.
- [26] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.
- [27] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.
- [28] G. Wahba. Support vector machines, reproducing kernel Hilbert spaces and the randomized gacv. Technical report no. 984, Department of Statistics, University of Wisconsin, Madison, WI 53706, 1997. <ftp://ftp.stat.wisc.edu/pub/wahba/index.html>.

Refining Initial Points for K-Means Clustering

P. S. Bradley

Computer Sciences Department
University of Wisconsin
Madison, WI 53706, USA
paulb@cs.wisc.edu

Usama M. Fayyad

Microsoft Research
Redmond, WA 98052, USA
fayyad@microsoft.com
<http://research.microsoft.com/~fayyad>

Abstract

Practical approaches to clustering use an iterative procedure (e.g. K-Means, EM) which converges to one of numerous local minima. It is known that these iterative techniques are especially sensitive to initial starting conditions. We present a procedure for computing a refined starting condition from a given initial one that is based on an efficient technique for estimating the modes of a distribution. The refined initial starting condition allows the iterative algorithm to converge to a "better" local minimum. The procedure is applicable to a wide class of clustering algorithms for both discrete and continuous data. We demonstrate the application of this method to the popular K-Means clustering algorithm and show that refined initial starting points indeed lead to improved solutions. Refinement run time is considerably lower than the time required to cluster the full database. The method is scalable and can be coupled with a scalable clustering algorithm to address the large-scale clustering problems in data mining.

1. BACKGROUND

Clustering is an important area of application for a variety of fields including data mining [FPSU96], statistical data analysis [KR89,BR93], compression [ZRL97], and vector quantization. Clustering has been formulated in various ways in the machine learning [F87], pattern recognition [DH73,F90], optimization [BMS97,SI84], and statistics literature [KR89,BR93,B95,S92,S86]. The fundamental clustering problem is that of grouping together (clustering) data items which are similar to each other. The most general approach to clustering is to view it as a density estimation problem [S86, S92,BR93]. We assume that in addition to the observed variables for each data item, there is a *hidden*, unobserved variable indicating the "cluster membership" of the given data item. Hence the data is assumed to arrive from a *mixture model* and the mixing labels (cluster identifiers) are hidden. In general, a mixture model M having K clusters C_i , $i=1,\dots,K$, assigns a probability to a data point x as follows:

$$\Pr(x | M) = \sum_{i=1}^K W_i \cdot \Pr(x | C_i, M) \text{ where } W_i \text{ are called the}$$

mixture weights. Many methods assume that the number of clusters K is known or given as input.

The clustering optimization problem is that of finding parameters associated with the mixture model M (W_i and parameters of components C_i) which maximize the likelihood of the data given the model. The probability distribution specified by each cluster can take any form. The EM algorithm [DLR77, CS96] is a well-known technique for estimating the parameters in the general case. K-Means clustering is a popular method (historically also known as *Forgy's method* [F65] or *MacQueen's algorithm* [M67]). It is really a special case of EM that assumes:

- 1) Each cluster is modeled by a spherical Gaussian distribution;
- 2) Each data item is assigned to a single cluster;
- 3) The mixture weights (W_i) are equal.

Note that K-Means [DH73,F90] is defined over numeric (continuous-valued) data since it requires the ability to compute the mean. A discrete version of K-Means exists and is sometimes referred to as *harsh EM* [NH98]. The K-Means algorithm finds locally optimal solutions minimizing the sum of the L_2 distance squared between each data point and its nearest cluster center ("distortion") [BMS97,SI84], which is equivalent to a maximizing the likelihood given the assumptions listed above.

There are various approaches to solving the problem of determining (locally) optimal values of the parameters given the data. *Iterative refinement* approaches, which include EM and K-Means, are the most effective. The basic algorithm works as follows:

- 1) Initialize the model parameters to a current model;
- 2) Decide memberships of the data items to clusters, assuming that the current model is correct;
- 3) Re-estimate the parameters of the current model assuming that the data memberships obtained in 2) are correct, producing new model;
- 4) If current model and new model are sufficiently close to each other, terminate, else go to 2).

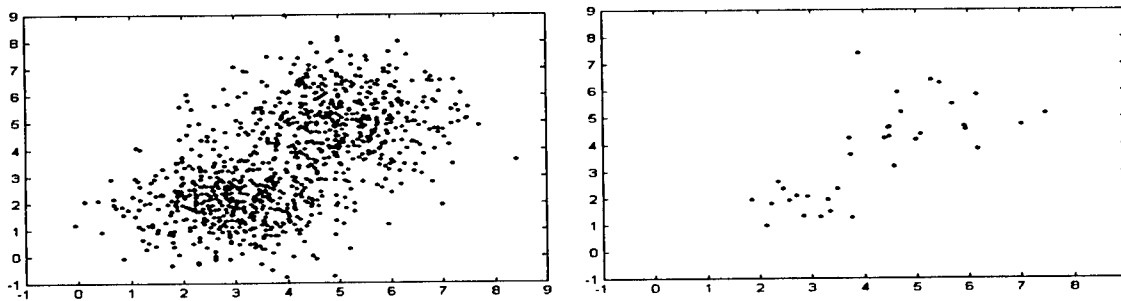


Figure 1. Two Gaussian bumps in 2-d: full sample versus small subsample.

We focus on the initialization step 1. Given the initial condition of step 1, the algorithms define a deterministic mapping from initial point to solution. Both the K-Means and EM algorithms converge finitely to a point (set of parameter values) that is locally maximal for the likelihood of the data given the model. The deterministic mapping means the locally optimal solution is sensitive to the initial point choice.

There is little prior work on initialization methods for clustering. According to [DH73] (p. 228):

"One question that plagues all hill-climbing procedures is the choice of the starting point. Unfortunately, there is no simple, universally good solution to this problem."

"Repetition with different random selections" [DH73] appears to be the *defacto* method. Most presentations do not address the issue of initialization or assume either user-provided or randomly chosen starting points [DH73, R92, KR89]. A recursive method for initializing the means by running K clustering problems is mentioned in [DH73]. A variant of this method consists of taking the mean of the entire data and then randomly perturbing it K times [TMCH97]. This method does not appear to be better than random initialization in the case of EM over discrete data [MH98]. In [BMS97], the values of initial means along any one of the d coordinate axes is determined by selecting the K densest "bins" along that coordinate.

Methods to initialize EM include K-Means solutions, hierarchical agglomerative clustering (HAC) [DH73, R92, MH98] and "marginal+noise" [TMCH97]. It was found that for EM over discrete data initialized with either HAC or "marginal+noise" showed no improvement over random initialization [MH98].

For the remainder of this paper we focus on the K-Means algorithm although the method can refine an initial point for other clustering algorithms. Our focus on K-Means is justified by the following: 1) it is a standard technique for clustering, used in a wide array of applications and even as way to initialize the more expensive EM clustering algorithm [B95, CS96, MH98]; 2) regardless of which clustering algorithm is being used, K-Means is employed internally by our initialization refinement method; 3) the

purpose of the paper is to illustrate the refinement procedure, not to evaluate a variety of clustering algorithms.

2. REFINING INITIAL CONDITIONS

We address the problem of initializing a general clustering algorithm, but limit our presentation of results to K-Means. Since no good method for initialization exists [MH98], we compare against the *defacto* standard method for initialization: randomly choosing an initial starting point. However, the method can be applied to any starting point provided.

A solution of the clustering problem is a parameterization of each cluster model. This parameterization can be performed by determining the *modes* (maxima) of the joint probability density of the data and placing a cluster centroid at each mode. Hence one clustering approach is to estimate the density and attempt to find the maxima ("bumps") of the estimated density function. Density estimation in high dimensions is difficult [S92], as is bump hunting [F90]. We propose a method, inspired by this procedure that refines the initial point to a point likely to be closer to the modes. The challenge is to perform refinement efficiently.

The basic heuristic is that severely subsampling the data will naturally bias the sample to representatives "near" the modes. In general, one cannot guard against the possibility of points from the tails appearing in the subsample. We have to overcome the problem that the estimate is fairly unstable due to elements of the tails appearing in the sample. Figure 1 shows data drawn from a mixture of two Gaussians (clusters) in 2-D with means at [3,2] and [5,5]. On the left is the full data set, on the right a small subsample is shown, providing information on the modes of the joint probability density function. Each of the points on the right may be thought of as a "guess" at the possible location of a mode in the underlying distribution. The estimates are fairly varied, but they certainly exhibit "expected" behavior. Worthy of note here is that good separation between the two clusters is achieved. This observation indicates that the solutions obtained by clustering over a small subsample may

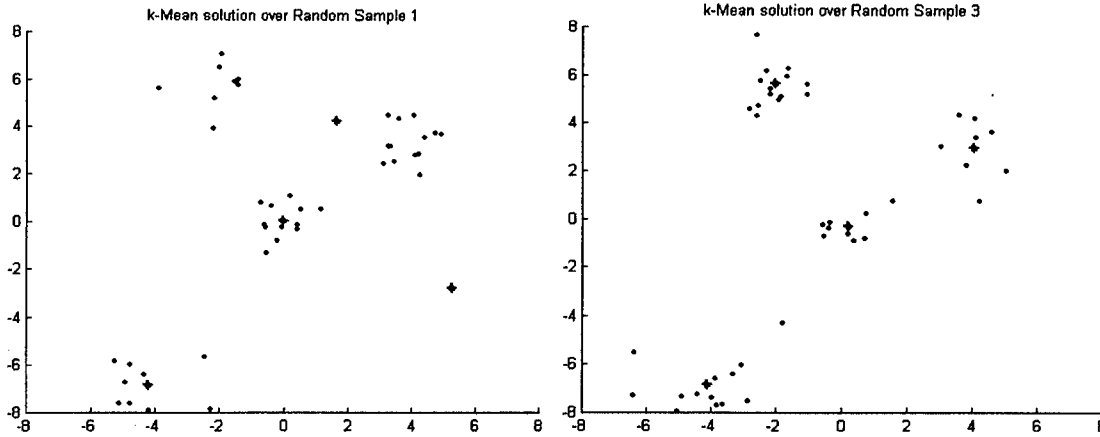


Figure 2: Result of clustering two different samples drawn from the same distribution, and initialized with the same starting point (produced solution indicated by '+').

provide good refined initial estimates of the true means, or centroids, in the data. However, this method often produces noisy estimates due to single small subsamples, especially in skewed distributions and high dimensions (Figure 2). This behavior is fairly common when clustering over small subsamples. In fact it is surprisingly frequent even in low dimensions using data from well-separated Gaussians¹. Figure 2 can also be used to illustrate the importance of the problem of having a good initial points. An initial cluster center attracting no data may remain empty (Figure 2, left), while a starting point with no empty clusters usually produces better solutions (right).

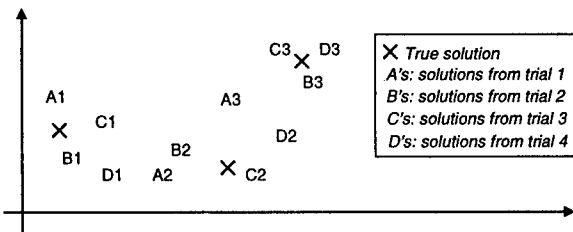


Figure 3. Multiple Solutions from Multiple Samples.

2.1 Clustering Clusters

In order to overcome the problem of noisy estimates, we employ the following procedure. Multiple subsamples, say J , are drawn and clustered independently producing J estimates of the true cluster locations. To avoid the noise associated with each of the J solutions, we employ a "smoothing" procedure. However, to "best" perform this smoothing, one needs to solve the problem of grouping the $K \cdot J$ points (J solutions, each having K clusters) into K groups in an "optimal" fashion. Figure 3 shows 4

solutions obtained for $K=3, J=4$. The "true" cluster means are depicted by "X". The A's show the 3 points obtained from the first subsample, B's second, C's third, and D's fourth. The problem then is determining that D1 is to be grouped with A1 but A2 should not be grouped with {A1, B1, C1, D1}.

2.2 The Refinement Algorithm

The refinement algorithm initially chooses J small random sub-samples of the data, $S_i, i=1, \dots, J$. The sub-samples are clustered via K-Means with the proviso that empty clusters at termination will have their initial centers re-assigned and the sub-sample will be re-clustered. The sets $CM_i, i=1, \dots, J$ are these clustering solutions over the sub-samples which form the set CM . CM is then clustered via K-Means initialized with CM_i producing a solution FM_i . The refined initial point is then chosen as the FM_i having minimal distortion over the set CM .

Clustering CM is a smoothing over the CM_i to avoid solutions "corrupted" by outliers included in the sub-sample S_i . The refinement algorithm takes as input: SP (initial starting point), $Data$, K , and J (number of small subsamples to be taken from $Data$):

Algorithm Refine($SP, Data, K, J$)

0. $CM = \emptyset$
1. For $i=1, \dots, J$
 - a. Let S_i be a small random subsample of $Data$
 - b. Let $CM_i = \text{KMeansMod}(SP, S_i, K)$
 - c. $CM = CM \cup CM_i$
2. $FMS = \emptyset$
3. For $i=1, \dots, J$
 - a. Let $FM_i = \text{KMeans}(CM_i, CM, K)$
 - b. Let $FMS = FMS \cup FM_i$
4. Let $FM = \text{ArgMin}\{\text{Distortion}(FM_i, CM)\}_{FM_i}$
5. Return (FM)

¹ In fact data from well-separated Gaussians in low-D are a "best-case" scenario for the behavior of a random sampling based approach. Note the idealized conditions: no noise, algorithm given the correct number of clusters K . With real-world data ideal conditions are difficult to achieve, hence the behavior is expected to be worse (and indeed it is).

We define the following functions called by the refinement algorithm: $KMeans()$, $KMeansMod()$ and $Distortion()$. $KMeans$ is simply a call to the classic K-Means algorithm taking: an initial starting point, dataset and the number of clusters K , returning a set of K d -dimensional vectors, the estimates of the centroids of the K clusters. $KmeansMod$ takes the same arguments as $KMeans$ (above) and performs the same iterative procedure as classic K-Means except for the following slight modification. When classic K-Means has converged, the K clusters are checked for membership. If any of the K clusters have no membership (which often happens when clustering over small subsamples), the corresponding initial estimates of the empty cluster centroids are set to data elements which are farthest from their assigned cluster center, and classic K-Means is called again from these new initial centroids.

The heuristic re-assignment is motivated by the following: if, at termination of K-Means, there are empty clusters then reassigning all empty clusters to points farthest from their respective centers decreases distortion most at this step. An example of clusters having zero membership is depicted in Figure 3 (left).

$Distortion$ takes set of K estimates of the means and the data set and computes the sum of squared distances of each data point to its nearest mean. This scalar measures the degree of fit of a set of clusters to the dataset. The K-Means algorithm terminates at a solution which is locally optimal for this distortion function [SI84]. The refinement process is illustrated in the diagram of Figure 4.

2.3 Computational Complexity and Scalability to Large Databases

The refinement algorithm is primarily intended to work on large databases. When working over small datasets (e.g. most data sets in the Irvine Repository), applying the classic K-Means algorithm from many different starting points is a feasible option. However, as database size increases (especially in dimensionality), efficient and accurate initialization becomes critical. A clustering session on a data set with many dimensions and tens of thousands or millions of records can take hours to days. In [BFR98], we present a method for scaling clustering to very large databases, specifically targeted at databases not fitting in RAM. We show that accurate clustering can be achieved with improved results over classic K-Means applied to an appropriately sized random subsample of the

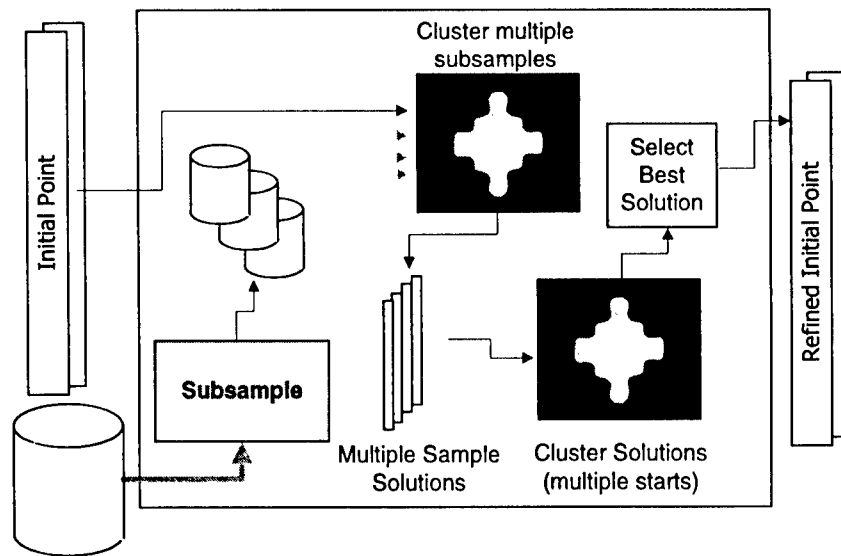


Figure 4. The Starting Point Refinement Procedure

database [BFR98]. Scalable clustering methods obviously benefit from better initialization.

Since our method works on very small samples of the data, the initialization is fast. For example, if we use sample sizes of 1% (or less) of the full dataset size, trials over 10 samples can be run in time complexity that is less than 10% of the time needed for clustering the full database. For very large databases, the initial sample becomes negligible in size.

If, for a data set D , a clustering algorithm requires $Iter(D)$ iterations to cluster it, then time complexity is $|D| * Iter(D)$. A small subsample $S \subseteq D$, where $|S| \ll |D|$, typically requires significantly fewer iteration to cluster. Empirically, it is safe to expect that $Iter(S) < Iter(D)$. Hence, given a specified budget of time that a user allocates to the refinement process, we simply determine the number J of subsamples to use in the refinement process. When $|D|$ is very large, and $|S|$ is a small proportion of $|D|$, refinement time is essentially negligible, even for large J .

Another desirable property of the refinement algorithm is that it easily scales to very large databases. The only memory requirement is to hold a small subsample in RAM. In the secondary clustering stage, only the solutions obtained from the J subsamples need to be held in RAM.

Note we assume that it is possible to obtain a *random sample* from a large-scale database. While this sounds simple, in reality this can be a challenging task. Unless one can guarantee that the records in a database are not ordered by some property, random sampling can be as expensive as scanning the entire database (using some scheme such as reservoir sampling, e.g. [J62]). Note that in a database environment, what one thinks of as a data

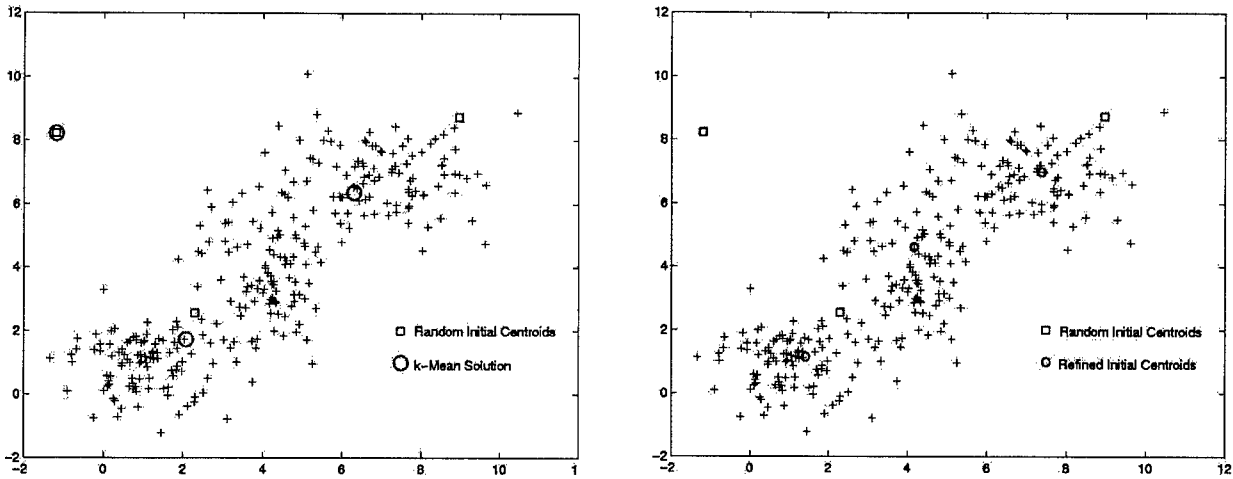


Figure 5: Left: K-Mean solution (large red circles) from random initial point (blue squares). Right: Refined initial point (red circles), random initial point (blue squares).

table (a view) may not exist as a physical table. The result of a query may involve joins, groupings, and sorts. In many cases database operations impose a special ordering on the result set, and “randomness” of the resulting *database view* cannot be assumed in general.

2.4 An Example

Figure 5 illustrates the sensitivity of K-Means solutions to initial conditions. Elements are sampled from three Gaussians in 2 dimensions. Note that the Gaussians in this case happen to be centered along a diagonal. The reason for this choice is that even as the dimensionality of the data goes higher, any 2 dimensional projection of the higher dimensional data will have this same form, making the data set *easy* for a visualization-based approach. Simply project the data to 2 dimensions, and the clusters reveal themselves. This is a rare property since, if the Gaussians are not aligned along the diagonal, any lower-dimensional projection may result in overlaps and separability in 2 dimensions is lost. The left figure shows a random starting point and the corresponding K-Means solution. The right figure shows the same initial random points and the result of the refinement procedure on this random initial point. Note that in this case the refined point is very close to the true solution. Running K-Means from the refined point converges to the true solution.

It is important to point out that this example is for illustrative purposes only. The interesting cases are high-dimensional data sets with more data items. Computational results indicate that the refinement method scales well to higher dimensions (100-D and more).

3. RESULTS ON SYNTHETIC DATA

3.1 Data Set Description

Synthetic data was created for dimension $d = 2, 3, 4, 5, 10, 20, 40, 50$ and 100 . For a given value of d , data was sampled from 10 Gaussians (hence $K=10$) with elements of their mean vectors (the true means) μ sampled from a uniform distribution on $[-5,5]$. Elements of the diagonal covariance matrices Σ were sampled from a uniform distribution on $[0.7,1.5]$. The number of data points sampled was chosen as 20 times the number of parameters estimated by K-Means. The $K=10$ Gaussians were not evenly weighted.

3.2 Experimental Methodology

The goal of this experiment is to evaluate how close the means estimated by classic K-Means are to the true Gaussian means generating the synthetic data. We compare 3 initializations:

1. *No Refinement*: random starting point chosen uniformly on the range of the data.
2. *Refinement ($J=10$)*: a starting point refined from (1) using our method. The size of the random subsamples being 10% of full dataset size and the number of subsamples taken being 10.
3. *Refinement ($J=1$)*: same as 2 but over a single random subsample of size 10%.

Once classic K-Means has computed a solution over the full dataset from any of the 3 initial points described above, the estimated means must be matched with the true Gaussian means in some optimal way prior to computing the distance between these estimated means the true Gaussian means. Let $\mu^l, l = 1, \dots, K$ be the K true Gaussian means and let $\bar{x}^l, l = 1, \dots, K$ be the K means estimated by classic K-Means over the full dataset. A “permutation” π is determined so that the following

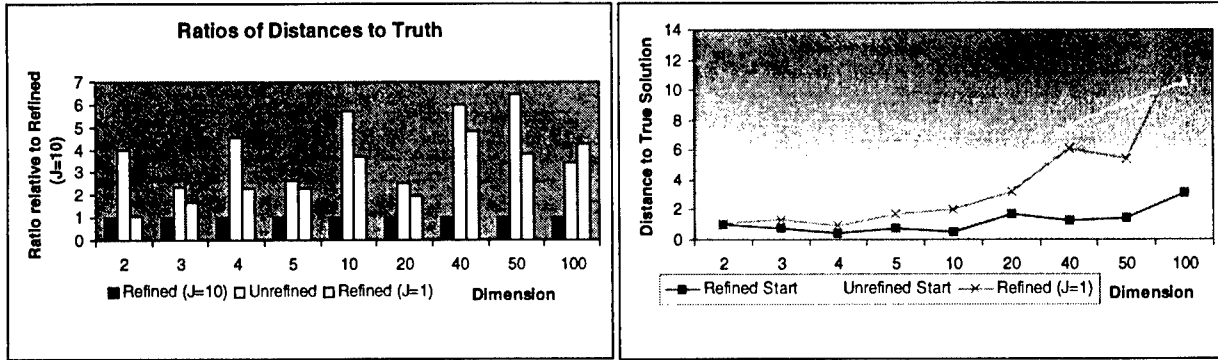


Figure 6. Comparing performance as dimensionality increases

quantity is minimized: $\sum_{l=1}^K \|\mu^l - \bar{x}^{\pi(l)}\|_2$. The “score” for

a solution computed by classic K-Means over the full dataset is simply the above quantity divided by K . This is the average distance between the true Gaussian means and those estimated by K-Means over the full dataset from a given initial starting point.

3.3 Experimental Results

Figure 6 summarizes results averaged over 10 random initial points determined uniformly on the range of the data. Note that the K-Means solution computed from “Refined (J=10)”, is consistently nearer to the true Gaussian means generating the dataset than the K-Means solution computed from either the random initial point or the “Refined (J=1)” initial point. On the left we summarize ratios of average distance to the true Gaussian means relative to the average distance for the classic K-Means solution computed from the refined initial point. Worth of note in these results are the following facts:

1. For dimensions 2-50, the refinement method (Refined (J=10)) *always* did better than the random starting point (Unrefined) and the point refined over 1 subsample (Refined (J=1)).
2. For dimension 100, in 9 of the 10 independent trials our refinement method did better than the random starting point.
3. Refiner solutions are between 2.34 ($d=3$) and 6.44 ($d=50$) times closer to the true Gaussian means than solutions from the random initial point and between 1.09 ($d=3$) and 4.80 ($d=50$) times closer than solution computed from “Refined (J=1)” initial point.

In one run, we did slightly worse. This explains the large variance number for 100 dimensions. If we exclude that one data point, the variance drops to within range of all other dimensions. The fact that the minimum ratio occurs for datasets with small dimensionality and the maximum ratio occurs for datasets with large dimensionality indicates the utility of the refinement algorithm for large-dimensional datasets.

4. RESULTS ON REAL-WORLD DATA

We present computational results on 2 classes of publicly available “real-world” datasets. We are primarily more interested in *large databases* -- hundreds of dimensions and tens of thousands to millions or records. It is for these data sets that our method exhibits the greatest value. The reason is simple: a clustering session on a large database is a time-consuming affair. Hence a refined starting condition can insure that the time investment pays off.

To illustrate this, we used a large publicly available data set available from Reuters News Service. The data is described in Section 4.2. We also wanted to demonstrate the refinement procedure using data sets from the UCI Machine Learning Repository. For the most part, we found that these data sets are *too easy*: they are low dimensional and have a very small number of records. With a small number of records, it is feasible to perform multiple restarts efficiently. Since the sample size is small to begin with, sub-sampling for initialization is not effective. Hence most of these data sets are not of interest to us. Nevertheless, we report on our general experience with them as well as detailed experience with one of these data sets to illustrate that the method we advocate is useful when applied to smaller data sets. We emphasize, however, that our refinement procedure is best suited for large-scale data. The refinement algorithm operates over small sub-samples of the database and hence run-times needed to determine a “good” initial starting point (which speeds the convergence on the full data set) are orders of magnitude less than the total time needed for clustering in a large-scale situation.

We note that it is very likely that the cluster labeling associated with many real-world databases do not correspond to the distortion measure minimized by K-Means.

4.1 Datasets from UCI ML Repository

We evaluated our method on several Irvine data sets. First we present results on the Image Segmentation data set, then we discuss the results over the other data sets.

Image Segmentation Data Set

This data set consists of 2310 data elements in 19 dimensions. Instances are drawn randomly from a database of 7 outdoor images (brickface, sky, foliage, cement, window, path, grass). Each of the 7 images is represented by 330 instances.²

Experimental Methodology

Random initial starting points were computed by sampling uniformly over the range of the data. We compare solutions achieved by the classic K-Means algorithm starting from: 1) random initial starting points, and 2) initial points refined by our method.

Once classic K-Means has converged, the "quality" of the solution must be determined. Unlike the case of synthetic data, we cannot measure *distance* to the true solution since "truth" is not known. However, we can use average class purity within each cluster as one measure of quality. The other measure, which is not dependent on a classification, is the distortion of the data given the clusters. Quality scoring methods are:

Information Gain: estimates the "amount of information" gained by clustering the database as measured by the reduction in class impurity within clusters. For a database with L known classes, let c^l be the number of data elements in class l where $l = 1, \dots, L$. Let m be the total number of data points in the database. The *Total Entropy*

of the database is:
$$\text{Total Entropy} = \sum_{l=1}^L \left(\frac{c^l}{m} \right) \log \left(\frac{c^l}{m} \right).$$

Upon convergence of the classic K-Means algorithm from a given initial starting point, the *Weighted Entropy* is computed over the given clustering as follows: Form the $K \times L$ cluster/class matrix C with the (i, j) -th element being the number of elements of class j belonging to cluster i . Notice that the clustering will completely recover the assigned classes if the cluster/class matrix has a permuted identity nonzero structure. Let CS_k be the size of the k -th cluster, then class entropy for the k -th cluster is

given by:
$$\text{ClusterEntropy}(k) = \sum_{l=1}^L \left(\frac{C_{k,l}}{CS_k} \right) \log \left(\frac{C_{k,l}}{CS_k} \right).$$

The weighted entropy of the entire clustering is given by:

$$\text{WeightedEntropy}(K) = \sum_{k=1}^K \left(\frac{CS_k}{M} \right) \text{ClusterEntropy}(k).$$

Information Gain = Total Entropy - Weighted Entropy(K).

Distortion: Given the K means estimated by the classic K-Means algorithm, the distortion value that we consider is simply the sum of the $L2$ distance squared between the data items and the mean of their assigned cluster. A smaller value for the distortion measure indicates that the model parameters (i.e. means) are a better fit to the database given the K-Means assumptions are true.

Results: Image Segmentation Database

Average information gain over 10 random initial points for classic K-Mean without refining the initial point was 0.3125 ± 0.3188 (\pm one standard deviation). Average information gain for K-Mean initialized from a refined ($J = 10$) starting point was 0.8195 ± 0.1458 . The amount of information gained on average by the solutions computed from the refined point was 2.6222 time that of the solution computed over the random initial point.

Furthermore, on average, solutions computed from the refined initial points ($J=10$) reduced distortion by 44.41% over solutions computed from random initial points.

4.2 Other Real World Datasets

We evaluated the refinement procedure on other data sets such as Fisher's IRIS, Star-Galaxy-Bright, etc. Because these data sets are very low dimensional and their sizes small, the majority of the results were of no interest.

Clustering these data sets from random initial points and from refined initial points led to approximately equal gain in entropy and equal distortion measures in most cases. We did observe, however, that *when a random starting point leads to a "bad" solution, then refinement indeed takes it to a "good" solution*. So in those (admittedly rare) cases, refinement does provide expected improvement. We use the Reuters information retrieval data set to demonstrate our method on a real and difficult clustering task.

Reuters Information Retrieval Data Set

The Reuters text classification database is derived from the original Reuters-21578 data set made publicly available as part of the Reuters Corpus, through available as part of the Reuters Corpus, through Reuters, Inc., Carnegie Group and David Lewis³. This data consists of 12,902 documents. Each document is a news article about some topic: e.g. *earnings, commodities, acquisitions, grain, copper*, etc... There are 119 categories, which belong to some 25 higher level categories (there is a hierarchy on categories). The Reuters database consists of word counts for each of the 12,902 documents. There are hundreds of thousands of words, but for purposes of our experiments we selected the 302 most frequently

² For a more detailed description of the data, see the the Irvine ML Data Repository at <http://www.ics.uci.edu/~mllearn/MLRepository.html>

³ See: <http://www.research.att.com/~lewis/reuters21578/README.txt> for more details on this data set.

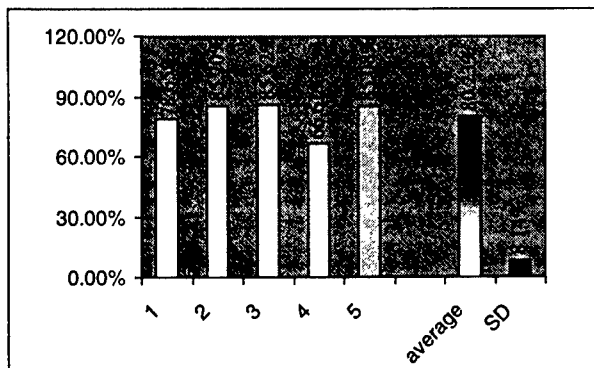


Figure 7: Results on Reuters Data from 5 Starting Points: percentage total distortion of refined solution relative to unrefined solution.

occurring words, hence each instance has 302 dimensions indicating the integer number of times the corresponding word occurs in the given document. Each document in the IR-Reuters database has been classified into one or more categories. We use $K=25$ for clustering purposes to reflect the 25 top-level categories. The task is then to find the best clustering given $K=25$.

Reuters Results

For this data set, because clustering the entire database requires a large amount of time, we chose to only evaluate results over 5 randomly chosen starting conditions. Results are shown in the chart of Figure 7. The chart shows a significant decrease in the total distortion measure. On average the distortion of a solution obtained by starting from a refined initial point was about 80% of the corresponding distortion obtained by clustering from the corresponding randomly chosen initial starting point.

Since each document belongs to a category (there are 119 categories), we can also measure the quality of the achieved by any clustering by measuring the gain in information about the categories that each cluster gives (i.e. pure clusters are informative). This is done in the same manner we measure entropy for the image segmentation dataset of Section 4.1. The quality of the clusters can be measured by the average category purity in each cluster. In this case the average information gain for the clusters obtained from the refined starting point was 4.13 times higher than the information gain obtained without refining the initial points. The information gain for the refined clustering was 0.071 with a standard deviation of 0.001. While the unrefined initial points resulted in an average information gain of 0.017 with a standard deviation equal to 0.011.

5. CONCLUDING REMARKS

A fast and efficient algorithm for refining an initial starting point for a general class of clustering algorithms has been presented. The refinement algorithm operates

over small subsamples of a given database, hence requiring a small proportion of the total memory needed to store the full database and making this approach very appealing for large-scale clustering problems. The procedure is motivated by the observation that subsampling can provide guidance regarding the location of the modes of the joint probability density function assumed to have generated the data. By initializing a general clustering algorithm near the modes, not only are the true clusters found more often, but it follows that the clustering algorithm will iterate fewer times prior to convergence. This is very important as the clustering methods discussed here require a full data-scan at each iteration and this may be a costly procedure in a large-scale setting.

Computational results on synthetic Gaussian data indicate that solutions computed by the K-Means algorithm from the refined initial points are superior to the random initial starting points and to a point refined over a single random subsample. Results on the small real-world Image Segmentation data set indicate that the K-Means solution from the refined points provide twice as much "information" than the solutions computed from the random initial point. Furthermore, the average distortion is decreased by 9%. Computational results on the Reuters database of newswire stories in 300 dimensions indicate a drop in distortion by about 20%. Information gain was improved by a factor of 4.13 times on this data set.

We believe that our method's ability to obtain a substantial refinement over randomly chosen starting points is due in large part to our ability to avoid the empty clusters problem that plagues traditional K-Means. Since during refinement we reset empty clusters to far points and reiterate the K-Means algorithm, a starting point obtained from our refinement method is less likely to lead the subsequent clustering algorithm to a "bad" solution. Our intuition is confirmed by the empirical results.

The refinement method presented so far has been in the context of the K-Means algorithm. However, we note that the same method is easily generalized to other algorithms, and even to discrete data (on which means are not defined). The generalized method and its use for initializing the EM algorithm, along with empirical results, is presented in [FRB98b]. The key insight here is that if some algorithm ClusterA is being used to cluster the data, then ClusterA is also used to cluster the subsamples. The algorithm ClusterA will produce a *model*. The model is essentially described by its parameters. The parameters are in a continuous space. The stage which clusters the clusters (i.e. step 3 of the algorithm Refine in Section 2.2) remains as is; i.e. we use the K-Means algorithm in this step. The reason for using K-Means is that the goal at this stage is to find the

"centroid" of the models, and in this case the *harsh* membership assignment of K-Means is desirable.

Acknowledgements

We thank Cory Reina for help on implementation, debugging, and running results over the large datasets; Sue Dumais and Mehran Sahami for making the Reuters data set available to us; and Chris Meek for valuable comments on an earlier draft.

References

- [BR93] J. Banfield and A. Raftery, "Model-based gaussian and non-Gaussian Clustering", *Biometrics*, vol. 49: 803-821, pp. 15-34, 1993.
- [B95] C. Bishop, 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.
- [BMS97] P. S. Bradley, O. L. Mangasarian, and W. N. Street. 1997. "Clustering via Concave Minimization", in *Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.) pp 368-374, MIT Press, 1997.
- [BFR98] P. S. Bradley, U. Fayyad, and C. Reina, "Scaling Clustering Algorithms to Large Databases", To appear, *Proc. 4th International Conf. on Knowledge Discovery and Data Mining (KDD-98)*. AAAI Press, Aug. 1998.
- [CS96] P. Cheeseman and J. Stutz, "Bayesian Classification (AutoClass): Theory and Results", in [FPSU96], pp. 153-180. MIT Press, 1996.
- [DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM algorithm". *Journal of the Royal Statistical Society, Series B*, 39(1): 1-38, 1977.
- [DH73] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*. New York: John Wiley and Sons. 1973
- [FHS96] U. Fayyad, D. Haussler, and P. Stolorz. "Mining Science Data." *Communications of the ACM* 39(11), 1996.
- [FPSU96] Fayyad, U., G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.) *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.
- [FRB98] U. Fayyad, C. Reina, and P. S. Bradley, "Refining Initialization of Expectation Maximization Clustering Algorithms", To appear, *Proc. 4th International Conf. on Knowledge Discovery and Data Mining (KDD-98)*. AAAI Press, Aug. 1998.
- [F87] D. Fisher. "Knowledge Acquisition via Incremental Conceptual Clustering". *Machine Learning*, 2:139-172, 1987.
- [F65] E. Forgy, "Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications", *Biometrics* 21:768. 1965.
- [F90] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, San Diego, CA: Academic Press, 1990.
- [J62] Jones, "A note on sampling from a tape file". *Communications of the ACM*, vol 5, 1962.
- [KR89] L. Kaufman and P. Rousseeuw, 1989. *Finding Groups in Data*, New York: John Wiley and Sons.
- [M67] J. MacQueen, "Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Volume I, Statistics, L. M. Le Cam and J. Neyman (Eds.). University of California Press, 1967.
- [MH98] M. Meila and D. Heckerman, 1998. "An experimental comparison of several clustering methods", *Microsoft Research Technical Report MSR-TR-98-06*, Redmond, WA.
- [NH98] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants", in M. I. Jordan (Ed.), *Learning in Graphical Models*, Kluwer: 1998.
- [R92] E. Rasmussen, "Clustering Algorithms", in *Information Retrieval Data Structures and Algorithms*, Frakes and Baeza-Yates (Eds.), pp. 419-442, New Jersey: Prentice Hall, 1992.
- [S92] D. W. Scott, *Multivariate Density Estimation*, New York: Wiley. 1992
- [SI84] S. Z. Selim and M. A. Ismail, "K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality." *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 1, 1984.
- [S86] B.W. Silverman, *Density Estimation for Statistics and Data Analysis*, London: Chapman & Hall, 1986.
- [TMCH97] B. Thiesson, C. Meek, D. Chickering, and D. Heckerman, 1997. "Learning Mixtures of Bayesian Networks", *Microsoft Research Technical Report TR-97-30*, Redmond, WA.
- [ZRL97] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: A new data clustering algorithm and its applications", *Data Mining and Knowledge Discovery*, vol. 1, no. 2, 1997.

Finite-time Regret Bounds for the Multiarmed Bandit Problem

Nicolò Cesa-Bianchi
DSI, University of Milan
via Comelico 39,
I-20315 Milano, Italy
cesabian@dsi.unimi.it

Paul Fischer
Lehrstuhl Informatik II
Universität Dortmund
D-44221 Dortmund, Germany
paulf@goedel.informatik.uni-dortmund.de

Abstract

We show finite-time regret bounds for the multiarmed bandit problem under the assumption that all rewards come from a bounded and fixed range. Our regret bounds after any number T of pulls are of the form $a + b \log T + c \log^2 T$, where a , b , and c are positive constants not depending on T . These bounds are shown to hold for variants of the popular ε -greedy and Boltzmann allocation rules, and for a new simple deterministic allocation rule. Moreover, our results also apply to an extension of the basic bandit problem in which reward distributions can depend, to some extent, from previous pulls and observed rewards. Finally, we discuss the empirical performance of our algorithms with respect to specific choices of the reward distributions.

1 INTRODUCTION

One of the fundamental issues in reinforcement learning is the exploration versus exploitation dilemma, whose simplest instance is, perhaps, the bandit problem. In its most basic formulation, a bandit problem is a set of N (with $N > 1$) gambling machines. When a machine is played (i.e., the “arm” of a bandit is pulled) it delivers a reward, which we assume here to be a number from a fixed and bounded real interval. A crucial feature is that each reward is an independent random variable. Moreover, all rewards delivered by the same machine are identically distributed according to some unknown and fixed law (note, however, that different machines may have different reward distributions). The goal of the player in the optimality model considered here is to minimize its regret, that is, the difference

between the expected total reward gained in a sequence of T plays and the expected total reward one could gain by playing T times any machine with maximum expected reward. The exploration versus exploitation dilemma is now clear: the player must trade-off the need to sample different machines, in order to compute reliable estimates of their expected reward, with the need of exploiting the machine with the highest current reward estimate, in order to keep the regret as low as possible throughout the sequence of plays.

A strategy for the player, also called “adaptive allocation rule”, is a method for selecting the arm to pull at each time t based on the rewards obtained during the previous $t - 1$ pulls. The classical result of Lai and Robbins [14] states that, asymptotically, the regret of any player strategy must be $\Omega(\log T)$, provided that the reward distributions satisfy some mild assumptions. In the same paper, Lai and Robbins also propose a general adaptive allocation rule that, whenever the reward distributions belong to some known parametric family, yields the optimal asymptotical regret $O(\log T)$ — see [1, 13] and references therein for extensions of these results. In this paper we show that simple variants of the popular ε -greedy and Boltzmann heuristics (see [11, 16] for a review of heuristics for the bandit problem) achieve a regret of the form $a + b \log T + c \log^2 T$ for all T (where a , b , and c are positive constants) when a lower bound on the difference between the highest and second-highest expected reward is known in advance. We also prove that the same regret bound holds for a new deterministic allocation rule. Our results do not require any further knowledge about the distributions of rewards and hold for any set of distributions with bounded rewards. Finally, our bounds apply, without modification, to a relaxed variant of the bandit problem, where the reward distributions can adversarially change after each play provided that each reward expectation is kept fixed.

2 DEFINITIONS AND NOTATION

Fix a positive integer $N > 1$. The N -armed bandit problem (with bounded rewards) is a collection of N random processes $\{X_{j,t} : t = 1, 2, \dots\}$, $j = 1, \dots, N$, satisfying

$0 \leq X_{j,t} \leq 1$ (this choice for the range of rewards is not crucial, by an appropriate scaling of the regret bounds any other bounded real interval would work). Each $X_{j,t}$ represents the random reward a player can obtain by pulling arm j at time t . An *adaptive allocation rule* for the N -armed bandit problem is an algorithm that, at each time t , chooses the index $I_t \in \{1, \dots, N\}$ of the next arm to pull based on the sequence

$$I_1, X_{I_1,1}, \dots, I_{t-1}, X_{I_{t-1},t-1}$$

of past pulls and observed rewards. We will also investigate *randomized* allocation rules, whose behavior depends on an additional internal random source. The (expected) *regret* after the first T pulls of the allocation rule that pull arms I_1, \dots, I_T is

$$\max_{1 \leq j \leq N} \mathbf{E} \left[\sum_{t=1}^T (X_{j,t} - X_{I_t,t}) \right]. \quad (1)$$

Here the expectation $\mathbf{E}[\cdot]$ is understood with respect to the stochastic generation of rewards and, for randomized rules, also with respect to the internal randomization of the rule.

In the standard formulation of the bandit problem it is assumed that the rewards delivered by the arms are independent random variables $X_{j,t}$ with stationary means μ_j , for $t = 1, 2, \dots$ and $j = 1, \dots, N$. All of our results will hold under this assumption. Moreover, all of our results will also hold under the weaker assumption that $\mathbf{E}[X_{j,t} | \mathcal{F}_{t-1}] = \mu_j$ for each t and j , where \mathcal{F}_{t-1} denotes the σ -field generated by random variables $I_1, X_{I_1,1}, \dots, I_{t-1}, X_{I_{t-1},t-1}$. In other words the distribution of each new random reward can depend in an adversarial way on the previous pulls and observed rewards, as long as its mean is kept fixed.

Throughout the paper, without loss of generality assume that $\mu_1 > \mu_j$ for all $j = 2, \dots, N$ and let $\Delta(\mu_1, \dots, \mu_N) = \min_{2 \leq j \leq N} (\mu_1 - \mu_j)$. Furthermore, let $\Delta_j = \mu_1 - \mu_j$ for each $j = 1, \dots, N$ and let $D = \sum_{j=1}^N \Delta_j$. Our allocation rules have an input parameter $d > 0$ and our regret bounds hold only if $0 < d \leq \Delta(\mu_1, \dots, \mu_N)$, where μ_1, \dots, μ_N are the unknown problem parameters. Furthermore, our bounds grow like $\Omega(1/d^2)$, so d should be chosen as close as possible to $\Delta(\mu_1, \dots, \mu_N)$. However, if an arbitrary value of d is fed to the allocation rules described in Section 3, we can still prove some weaker form of regret bound.

Finally, we use \ln for the natural logarithm and \log for the base 2 logarithm.

3 REGRET BOUNDS

Many heuristics for the bandit problem assign to each arm i a probability of being pulled that is proportional to the current reward estimate for arm i . A popular example is the *Boltzmann Exploration* (BE) heuristic (see [3] and references therein). This allocation rule, at each time t , draws the arm to pull according to the exponential distribution

$e^{\hat{\mu}_{i,t-1}/\tau} / Z_t$ for $i = 1, \dots, N$, where $\hat{\mu}_{i,t-1}$ is the current estimate of the expected reward for arm i , the quantity $\tau > 0$ is a “temperature” parameter, and Z_t is a normalization factor. Note that, for $\tau \rightarrow 0$, BE reduces to the greedy rule always choosing to pull the arm with the highest current reward estimate. On the other hand, for $\tau \rightarrow \infty$ arms are pulled independently and uniformly at random. Similarly to the Simulated Annealing optimization method [12, 17], one can obtain empirical convergence to the best arm by letting $\tau = \tau_t$ monotonically decrease to 0 according to some “cooling schedule”. A natural question is then whether there exists a cooling schedule which provably yields convergence to the best arm. We now introduce a variant of BE, called SOFTMIX, for which we can construct such an “optimal” cooling schedule. The algorithm SOFTMIX (see Figure 1) uses the exponential distribution mixed with the uniform distribution.¹ This is equivalent to saying that, at each time t , we flip a biased coin to decide whether the next arm to pull should be drawn from the exponential distribution (with a prescribed finite temperature value) or from the uniform distribution (corresponding to the exponential distribution with infinite temperature). We use γ_t to denote the bias (which we also call *mixing coefficient*) of the coin. A crucial aspect is that both the temperature parameter τ_t and the mixing coefficient γ_t decrease with t following a schedule chosen so to minimize the regret bound in our analysis. More precisely, we set $\gamma_t = \Theta(\ln(t)/t)$ and $\tau_t = \Theta(1/\ln(t))$. For notational convenience, τ_t is replaced by an “inverse temperature” parameter $1/\eta_t$. The performance of SOFTMIX is analyzed in the following result.

Theorem 3.1 *For all integers $N > 1$ and for all N -armed bandit problems with parameters μ_1, \dots, μ_N , if $0 < d \leq \Delta(\mu_1, \dots, \mu_N)$ then, for all $T \geq 1$, the regret after the first T pulls of the randomized allocation rule SOFTMIX described in Figure 1 is at most*

$$\frac{D}{d^2} \left(8 \ln \frac{5N}{d^2} + \frac{5}{2} \ln^2 T \right) + D \ln T.$$

Recall that in the “zero temperature limit”, i.e. when the temperature parameter τ approaches 0, BE becomes greedy: at each time t , the arm i maximizing the reward estimate $\hat{\mu}_{i,t-1}$ gets pulled (ties are broken at random). The obvious flaw in this strategy is that an early unlucky sampling of some suboptimal arm might prevent the optimal arm from being sampled enough. A more successful variant of the greedy rule is the so-called ε -greedy heuristic (see, e.g., [18]). At each time t , this strategy pulls with probability $1 - \varepsilon$ any arm with the highest reward estimate and pulls with probability ε a randomly chosen arm. Now note that the zero temperature limit of SOFTMIX (attained when the inverse temperature parameter η_t approaches infinity) corresponds to the ε -greedy heuristic with the setting

¹The same mix was used in [2]. However, here the mixing coefficient is dynamically adapted to minimize the regret uniformly over time, whereas in [2] it was kept constant.

Randomized allocation rule: SOFTMIX.**Input:** Real number $0 < d < 1$.**Initialization:** Define sequences $\gamma_t \in (0, 1]$ and $\eta_t > 0$, $t = 1, 2, \dots$, by

$$\gamma_t = \begin{cases} \min \left\{ 1, \frac{5N}{d^2} \frac{\ln(t-1)}{t-1} \right\} & \text{if } t > 2, \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

and

$$\eta_t = \frac{1}{N/\gamma_t + 1} \ln \left(1 + \frac{d(N/\gamma_t + 1)}{2N/\gamma_t - d^2} \right). \quad (3)$$

Let $\hat{s}_j = 0$ for $j = 1, \dots, N$.**Loop:** For each $t = 1, 2, \dots$

- Pull an arm drawn from the distribution $\{P_{1,t}, \dots, P_{N,t}\}$, where

$$P_{i,t} = (1 - \gamma_t) \frac{e^{\eta_t \hat{s}_i}}{\sum_{j=1}^N e^{\eta_t \hat{s}_j}} + \frac{\gamma_t}{N}. \quad (4)$$

- Let i_t be the index of the pulled arm and x_{i_t} the reward obtained. Add $x_{i_t}/P_{i_t,t}$ to \hat{s}_{i_t} .

Figure 1: Description of the randomized allocation rule SOFTMIX.

$\varepsilon = \gamma_t$. This observations suggests that the two allocation rules might have similar behaviour, especially when t is large. The experiments of Section 6 (see Figure 4) confirm this conclusion: SOFTMIX has a better start but, for t large enough, the two algorithms exhibit the same behavior. On the other hand, we now state an upper bound on the regret of the γ_t -greedy heuristic identical to the one we proved for SOFTMIX. So, with respect to our analysis, the more sophisticated selection method used by SOFTMIX does not provide any extra benefit.

Theorem 3.2 *For all integers $N > 1$ and for all N -armed bandit problems with parameters μ_1, \dots, μ_N , if $0 < d \leq \Delta(\mu_1, \dots, \mu_N)$ then, for all $T \geq 1$, the regret after the first T pulls of the randomized allocation rule GREEDYMIX described in Figure 2 is at most*

$$\frac{D}{d^2} \left(8 \ln \frac{5N}{d^2} + \frac{5}{2} \ln^2 T \right) + D \ln T.$$

Some heuristics for the bandit problem, like the so-called “optimism in the face of uncertainty” exhibit a two-phase behaviour (see [11, Section 2.2.1] for a list of references). In the first phase exploration is favored; in the second phase exploitation takes over and the heuristic operates in a completely greedy way. By extending the initial explorative phase long enough one can make arbitrarily small (though not vanishing) the risk of converging to a suboptimal arm.

Randomized allocation rule: GREEDYMIX.**Input:** Real number $0 < d < 1$.**Initialization:** Define the sequence $\gamma_t \in (0, 1]$, $t = 1, 2, \dots$, by

$$\gamma_t = \begin{cases} \min \left\{ 1, \frac{5N}{d^2} \frac{\ln(t-1)}{t-1} \right\} & \text{if } t > 2, \\ 1 & \text{otherwise.} \end{cases}$$

Let $\hat{s}_j = 0$ for $j = 1, \dots, N$.**Loop:** For each $t = 1, 2, \dots$

- Let \mathcal{I} be the subset of arms such that, for each $i \in \mathcal{I}$, $\hat{s}_i = \max_{1 \leq j \leq N} \hat{s}_j$.
- With probability $1 - \gamma_t$ pull a random arm in \mathcal{I} , with probability γ_t pull a random arm.
- Let i_t be the index of the pulled arm and x_{i_t} the reward obtained. Add $x_{i_t}/Q_{i_t,t}$ to \hat{s}_{i_t} , where $Q_{j,t}$ is the probability of $i_t = j$ according to the rule above, that is

$$Q_{j,t} = \begin{cases} (1 - \gamma_t)/|\mathcal{I}| + \gamma_t/N & \text{if } j \in \mathcal{I}, \\ \gamma_t/N & \text{otherwise.} \end{cases} \quad (5)$$

Figure 2: Description of the randomized allocation rule GREEDYMIX.

We propose a new strategy, called ROUNDS, where a purely explorative phase is alternated with a purely exploitative phases. To guarantee a good bound on the regret, the length of the r -th exploitation is 2^r whereas the length of the exploration phases grows only linearly. The theoretical performance of ROUNDS (which is a deterministic allocation rule) turns out to be comparable to that of the randomized strategies SOFTMIX and GREEDYMIX. On the other hand, our experiments indicate that both randomized rules have an expected performance better than ROUNDS, especially for small values of T .

Theorem 3.3 *For all integers $N > 1$ and for all N -armed bandit problems with parameters μ_1, \dots, μ_N , if $0 < d \leq \Delta(\mu_1, \dots, \mu_N)$ then, for all $T \geq 1$, the regret after the first T pulls of the deterministic allocation rule ROUNDS described in Figure 3 is at most*

$$\left(\frac{2D \log(2N)}{d^2} + 1 \right) \lceil \log(T+1) \rceil + \frac{D}{2d^2} \lceil \log(T+1) \rceil^2.$$

Our results of Section 3 hold under the assumption that a lower bound $d > 0$ on the smallest difference $\mu_1 - \mu_j$, $j \neq 1$ is known. Arbitrary values of d , however, still allow to prove reasonable bounds on the regret. In fact, the regret bound is similar as before with an additional ΔT term, where Δ is the difference between μ_1 (the highest expected reward) and the smallest μ_i strictly bigger than $\mu_1 - d$. (If

Allocation rule: ROUNDS.

Input: Real number $0 < d < 1$.

Loop: For each round $r = 0, 1, \dots$

- Let $T_r = \lceil 2(\log_2(2N) + r)/d^2 \rceil$.
- For each arm $i = 1, \dots, N$: Pull the arm i for T_r times and let $s_{i,r}$ the total reward obtained.
- Let k be such that $s_{k,r} = \max_{1 \leq j \leq N} s_{j,r}$. Pull arm k for 2^r times.

Figure 3: Description of the deterministic allocation rule ROUNDS.

d not larger than the smallest difference $\mu_1 - \mu_j$ ($j \neq 1$), then Δ is 0 and we recover the previous bound.)

Corollary 3.4 For all integers $N > 1$, for all N -armed bandit problems with parameters μ_1, \dots, μ_N , and for all $T \geq 1$, the regret of both randomized allocation rules SOFTMIX and GREEDYMIX with input $d > 0$ is at most

$$\Delta_{i(d)}T + \frac{D(d)}{d^2} \left(8 \ln \frac{5N}{d^2} + 3 \ln^2 T \right)$$

where

$$\Delta_{i(d)} = \max_{\{i: \mu_i > \mu_1 - d\}} \Delta_i$$

and

$$D(d) = \sum_{\{j: \mu_j \leq \mu_1 - d\}} \Delta_j.$$

4 REMARKS

Unbiased estimates. The randomized allocation rules SOFTMIX and GREEDYMIX use a special kind of estimate, $\hat{s}_{i,t}/t$, for the expected reward of each arm i . If $P_{i,s}$ is the probability of pulling arm i at time s , then the reward estimate at time t for arm i is

$$\frac{\hat{s}_{i,t-1}}{t-1} = \frac{1}{t-1} \sum_{s=1}^{t-1} \hat{X}_{i,s} \quad (6)$$

where $\hat{X}_{i,s} = X_{i,s}/P_{i,s}$ if arm i was pulled at time s and $\hat{X}_{i,s} = 0$ otherwise. This estimate, which was previously used in [2] to solve a variant of the bandit problem substantially different from the one studied here, has the correct expectation μ_i for each arm i . In fact, we have

$$\mathbf{E} [\hat{X}_{i,s}] = \mathbf{E} \left[\frac{X_{i,s}}{P_{i,s}} P_{i,s} + 0(1 - P_{i,s}) \right] = \mu_i.$$

We could not prove our results for a different choice of the reward estimates.

Cooling schedule. In order to compare the inverse temperature parameter η_t of SOFTMIX with the temperature parameter τ_t of BE, in Section 3 we said that the sequence of values η_t for $t = 1, 2, \dots$ corresponds to a cooling schedule $\tau_t = \Theta(1/\ln t)$. To see why, recall that the expression for the probability of drawing arm i in BE has $\hat{\mu}_{i,t-1}/\tau_t$ at the exponent, where $\hat{\mu}_{i,t-1}$ is the current estimate of the expected reward for arm i . The corresponding exponent for the probability of drawing arm i in SOFTMIX is $\hat{s}_{i,t-1}\eta_t$ (we are disregarding the contributions of the factor $1 - \gamma_t$ and of the term γ_t/N , both negligible for t large). As, by (6), SOFTMIX's reward estimate is $\hat{s}_{i,t-1}/(t-1)$, we get that $\hat{\mu}_{i,t-1}/\tau_t = \hat{\mu}_{i,t-1}/(t-1)\eta_t$. Hence $\tau_t = 1/((t-1)\eta_t)$. Asymptotically, the quantity $(t-1)\eta_t$ shows a logarithmic growth

$$\lim_{t \rightarrow \infty} \frac{(t-1)\eta_t}{\ln(t-1)} = \frac{5}{d^2} \ln(1 + d/2).$$

Recall that the idea of BE with cooling is borrowed from the Simulated Annealing (SA) optimization method. A remarkable fact is that the cooling schedule necessary and sufficient for convergence (with probability 1) of SA to the global optimum is also $\Theta(1/\ln t)$, as shown in [7].

Instantaneous regret bounds. The proof of Theorems 3.1 and 3.2 also yields bounds on the instantaneous regret of both SOFTMIX and GREEDYMIX. In particular, for all $t > \lceil (8N/d^2) \ln(5N/d^2) \rceil$,

$$\mathbf{E} [X_{1,t} - X_{i_t,t}] \leq \frac{D}{t-1} + \frac{5D \ln(t-1)}{d^2(t-1)}$$

where i_t is the arm pulled at time t by any allocation rule between SOFTMIX and GREEDYMIX.

Similarity of the regret bounds. The dominant term in the regret bound for the three allocation rules considered here is, recalling that $D = O(N)$, of the order of $(N/d^2) \log^2 T$. This similarity is not by accident. In Section 5 we show how the regret of both SOFTMIX and GREEDYMIX can be reduced to the expectation of the product of moment generating functions for certain random variables — see (9) and (15). This product is bounded term by term using Taylor expansion of the exponential function. For the deterministic rule ROUNDS, we control the accuracy of the worst current reward estimate via standard Hoeffding bounds. As Hoeffding bounds are again proven through Taylor bounds on the moment generating function, we get similar rates for the regret. Observe that, in both cases, the Taylor expansion heavily relies on the boundedness of the rewards.

Interval estimation method. Another popular allocation rule, which works very well in empirical trials, is Kaelbling's interval estimation method [10]. This method operates by computing upper bound estimates $u_{i,t}$ for the expected reward μ_i of each arm i satisfying

$$\mathbf{P} \{ \mu_i \geq u_{i,t} \} = \theta \quad (7)$$

for some parameter $\theta > 0$. The interval estimation rule picks, at each time t , the arm i maximizing $u_{i,t}$ for a fixed value of θ . Clearly, to compute $u_{i,t}$ satisfying (7) one needs some information on the reward distributions. For Bernoulli bandits (with rewards chosen in $\{0, 1\}$ for each arm), one can use the Normal approximation to the binomial distribution and then apply standard formulae to compute the quantities $u_{i,t}$. For unknown reward distributions, which is the case of our setup, one must resort to general estimates in much the same way we used Hoeffding bounds to control the regret of ROUNDS.

5 PROOFS

We will make use of the following fact which can be easily verified by Taylor expansion of the exponential function (a proof can be found in [15, page 155]).

Fact 5.1 For every real $c > 0$, define the function ϕ_c on the positive reals by $\phi_c(z) = (e^{cz} - 1 - cz)/c^2$. Then, for every $y \leq c$ and every $z > 0$, $e^{zy} \leq 1 + zy + \phi_c(z)y^2$.

Proof of Theorem 3.1. Let $P_{j,t} = \mathbf{P}\{I_t = j \mid \mathcal{F}_{t-1}\}$ be defined as in (4). Recall that we are assuming $\mu_1 = \max_{1 \leq i \leq N} \mu_i$. We rewrite the regret (1) as follows

$$\begin{aligned} & \max_{1 \leq j \leq N} \mathbf{E} \left[\sum_{t=1}^T (X_{j,t} - X_{I_t,t}) \right] \\ &= \mathbf{E} \left[\sum_{t=1}^T (X_{1,t} - X_{I_t,t}) \right] \\ &= \sum_{t=1}^T \mathbf{E} [X_{1,t} - X_{I_t,t}] \\ &= \sum_{t=1}^T \mathbf{E} [\mathbf{E} [X_{1,t} - X_{I_t,t} \mid \mathcal{F}_{t-1}]] \quad (8) \\ &= \sum_{t=1}^T \mathbf{E} \left[\sum_{j=2}^N \Delta_j \mathbf{P}\{I_t = j \mid \mathcal{F}_{t-1}\} \right] \\ &= \sum_{t=1}^T \sum_{j=2}^N \Delta_j \mathbf{E} [P_{j,t}]. \end{aligned}$$

In (8) the inner conditional expectation is understood with respect to both the random choice of I_t and the random realization of the reward $X_{I_t,t}$. The outer expectation simply averages over the past $t-1$ pulls and obtained rewards. Define random variables

$$\hat{X}_{j,t} = \begin{cases} X_{j,t}/P_{j,t} & \text{if } I_t = j, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\hat{X}_{j,t} \leq 1/P_{j,t} \leq N/\gamma_t$, a fact which we use several times throughout this section. We have

$$\mathbf{E} [P_{i,t}] = (1 - \gamma_t) \mathbf{E} \left[\frac{e^{\eta_t \sum_{s=1}^{t-1} \hat{X}_{i,s}}}{\sum_{j=1}^N e^{\eta_t \sum_{s=1}^{t-1} \hat{X}_{j,s}}} \right] + \frac{\gamma_t}{N}$$

$$\begin{aligned} & \leq (1 - \gamma_t) \mathbf{E} \left[\frac{e^{\eta_t \sum_{s=1}^{t-1} \hat{X}_{i,s}}}{e^{\eta_t \sum_{s=1}^{t-1} \hat{X}_{1,s}}} \right] + \frac{\gamma_t}{N} \\ &= (1 - \gamma_t) \mathbf{E} \left[e^{\eta_t \sum_{s=1}^{t-1} (\hat{X}_{i,s} - \hat{X}_{1,s})} \right] + \frac{\gamma_t}{N} \\ &= (1 - \gamma_t) \mathbf{E} \left[\prod_{s=1}^{t-1} \mathbf{E} \left[e^{\eta_t (\hat{X}_{i,s} - \hat{X}_{1,s})} \mid \mathcal{F}_{s-1} \right] \right] + \frac{\gamma_t}{N} \\ &\leq e^{-(t-1)\eta_t \Delta_i} \mathbf{E} \left[\prod_{s=1}^{t-1} \mathbf{E} \left[e^{\eta_t \hat{Z}_{i,s}} \mid \mathcal{F}_{s-1} \right] \right] + \frac{\gamma_t}{N} \quad (9) \end{aligned}$$

$$\text{for } \hat{Z}_{i,t} = \hat{X}_{i,t} - \hat{X}_{1,t} + \Delta_i.$$

In the last step we multiplied and divided by the same quantity $e^{(t-1)\eta_t \Delta_i}$ and then we dropped the factor $0 < 1 - \gamma_t \leq 1$. In view of bounding each factor $\mathbf{E} [e^{\eta_t \hat{Z}_{i,s}} \mid \mathcal{F}_{s-1}]$ via Fact 5.1, first we compute the conditional expectation for each s ,

$$\begin{aligned} & \mathbf{E} [\hat{Z}_{i,s} \mid \mathcal{F}_{s-1}] \\ &= \mathbf{E} [\hat{X}_{i,s} \mid \mathcal{F}_{s-1}] - \mathbf{E} [\hat{X}_{1,s} \mid \mathcal{F}_{s-1}] + \Delta_i \\ &= \mu_i - \mu_1 + \Delta_i = 0. \end{aligned}$$

Second, observing that γ_s is positive and nonincreasing in s , we get $\hat{Z}_{i,s} = \hat{X}_{i,s} - \hat{X}_{1,s} + \Delta_i \leq N/\gamma_t + 1$. Third, using the same observation, we also bound the conditional variance for each s as follows.

$$\begin{aligned} \mathbf{E} [\hat{Z}_{i,s}^2 \mid \mathcal{F}_{s-1}] &= \mathbf{E} \left[(\hat{X}_{i,s} - \hat{X}_{1,s})^2 \mid \mathcal{F}_{s-1} \right] \\ &\quad + \Delta_i^2 + 2\Delta_i(\mu_i - \mu_1) \\ &= \mathbf{E} \left[(\hat{X}_{i,s} - \hat{X}_{1,s})^2 \mid \mathcal{F}_{s-1} \right] - \Delta_i^2 \\ &= \mathbf{E} [\hat{X}_{i,s}^2 \mid \mathcal{F}_{s-1}] + \mathbf{E} [\hat{X}_{1,s}^2 \mid \mathcal{F}_{s-1}] \\ &\quad - 2\mathbf{E} [\hat{X}_{1,s} \hat{X}_{i,s} \mid \mathcal{F}_{s-1}] - \Delta_i^2 \\ &= \mathbf{E} [\hat{X}_{i,s}^2 \mid \mathcal{F}_{s-1}] + \mathbf{E} [\hat{X}_{1,s}^2 \mid \mathcal{F}_{s-1}] - \Delta_i^2 \\ &\quad \text{as } \hat{X}_{1,s} = 0 \text{ or } \hat{X}_{i,s} = 0 \\ &= \mathbf{E} \left[\frac{X_{i,s}^2}{P_{i,s}^2} P_{i,s} + 0(1 - P_{i,s}) \mid \mathcal{F}_{s-1} \right] \\ &\quad + \mathbf{E} \left[\frac{X_{1,s}^2}{P_{1,s}^2} P_{1,s} + 0(1 - P_{1,s}) \mid \mathcal{F}_{s-1} \right] - \Delta_i^2 \\ &\leq \frac{1}{P_{i,s}} + \frac{1}{P_{1,s}} - \Delta_i^2 \leq \frac{2N}{\gamma_t} - \Delta_i^2 \\ &\quad \text{as } P_{i,s} \geq \gamma_t/N \text{ for } i = 1, \dots, N. \end{aligned}$$

Hence, applying Fact 5.1 with $c = N/\gamma_t + 1$ and $z = \eta_t$ we find that

$$\mathbf{E} [e^{\eta_t \hat{Z}_{i,s}} \mid \mathcal{F}_{s-1}] \leq \mathbf{E} [1 + \eta_t \hat{Z}_{i,s} + \hat{Z}_{i,s}^2 \phi_c(\eta_t) \mid \mathcal{F}_{s-1}]$$

$$\begin{aligned} &\leq 1 + \left(\frac{2N}{\gamma_t} - \Delta_i^2 \right) \phi_c(\eta_t) \\ &\leq \exp \left(\phi_c(\eta_t) \left(\frac{2N}{\gamma_t} - \Delta_i^2 \right) \right) \end{aligned}$$

for all $s = 1, \dots, t-1$. Thus

$$\begin{aligned} \mathbf{E}[P_{i,t}] &\leq e^{-(t-1)\eta_t \Delta_i} \mathbf{E} \left[\prod_{s=1}^{t-1} \exp \left(\phi_c(\eta_t) \left(\frac{2N}{\gamma_t} - \Delta_i^2 \right) \right) \right] \\ &\quad + \frac{\gamma_t}{N} \\ &= \exp \left(-(t-1) \left(\eta_t \Delta_i - \phi_c(\eta_t) \frac{2N}{\gamma_t} + \phi_c(\eta_t) \Delta_i^2 \right) \right) \\ &\quad + \frac{\gamma_t}{N} \\ &\leq \exp \left(-(t-1) \left(\eta_t d - \phi_c(\eta_t) \frac{2N}{\gamma_t} + \phi_c(\eta_t) d^2 \right) \right) \\ &\quad + \frac{\gamma_t}{N} \end{aligned} \quad (10)$$

where (10) is shown using the assumption $d \leq \Delta(\mu_1, \dots, \mu_N)$. By letting $(t-1)d = K$ and $(t-1)(2N/\gamma_t - d^2) = \sigma^2$ the term at the exponent in (10) becomes

$$-K\eta_t + \phi_c(\eta_t)\sigma^2. \quad (11)$$

Rewriting η_t in the simpler form $(1/c) \ln(1 + cK/\sigma^2)$, replacing ϕ_c with its definition, and using the elementary inequality $\ln(1+x) \geq 2x/(2+x)$, $x \geq 0$, we get that the quantity in (11) is smaller or equal to $-K^2/(2\sigma^2 + cK)$. Hence, plugging back in the original expression for K , σ^2 , and c , and simplifying the $t-1$ factor, we find that (10) is smaller or equal to

$$\begin{aligned} &\exp \left(-\frac{(t-1)d^2}{4N/\gamma_t - 2d^2 + d(N/\gamma_t + 1)} \right) + \frac{\gamma_t}{N} \\ &\leq \exp \left(-\frac{(t-1)d^2\gamma_t}{5N} \right) + \frac{\gamma_t}{N}. \end{aligned} \quad (12)$$

Now, for $t > T_0 = \lceil (8N/d^2) \ln(5N/d^2) \rceil$, we have that

$$\gamma_t = \frac{5N \ln(t-1)}{d^2(t-1)} < 1.$$

The choice of γ_t balances the contributions of the terms in the right-hand side of (12). Thus, for all $t > T_0$ and all $j \in \{2, \dots, N\}$, we can further bound the right-hand side of (12) as follows

$$\exp \left(-\frac{(t-1)d^2\gamma_t}{5N} \right) + \frac{\gamma_t}{N} \leq \frac{1}{t-1} + \frac{5 \ln(t-1)}{d^2(t-1)}.$$

For $t \leq T_0$ the mixing coefficient γ_t is 1. Hence, for each such t , the regret is Δ_j with probability $1/N$ for each j . Piecing everything together we obtain the desired bound

on the regret

$$\begin{aligned} &\mathbf{E} \left[\sum_{t=1}^T (X_{1,t} - X_{i_t,t}) \right] \\ &= \sum_{j=1}^N \Delta_j \left(\sum_{t=1}^{T_0} \mathbf{E}[P_{j,t}] + \sum_{t=T_0+1}^T \mathbf{E}[P_{j,t}] \right) \\ &= \sum_{j=1}^N \Delta_j \left(\sum_{t=1}^{T_0} \frac{1}{N} + \sum_{t=T_0+1}^T \mathbf{E}[P_{j,t}] \right) \\ &\leq \sum_{j=1}^N \Delta_j \left(\frac{8}{d^2} \ln \frac{5N}{d^2} + \sum_{t=T_0+1}^T \frac{1}{t-1} \right. \\ &\quad \left. + \frac{5}{d^2} \sum_{t=T_0+1}^T \frac{\ln(t-1)}{t-1} \right) \\ &\leq \sum_{j=1}^N \Delta_j \left(\frac{8}{d^2} \ln \frac{5N}{d^2} + \ln T + \frac{5}{d^2} \int_1^T \frac{\ln x}{x} dx \right) \\ &= \sum_{j=2}^N \Delta_j \left(\frac{8}{d^2} \ln \frac{5N}{d^2} + \ln T + \frac{5}{2d^2} \ln^2 T \right). \end{aligned}$$

Proof of Theorem 3.2. The regret (1) can be re-written as follows

$$\begin{aligned} &\max_{1 \leq j \leq N} \mathbf{E} \left[\sum_{t=1}^T (X_{j,t} - X_{I_t,t}) \right] \\ &= \mathbf{E} \left[\sum_{t=1}^T (X_{1,t} - X_{I_t,t}) \right] \\ &= \sum_{t=1}^T \mathbf{E}[X_{1,t} - X_{I_t,t}] \\ &= \sum_{t=1}^T \sum_{j=2}^N \Delta_j \mathbf{P}\{I_t = j\} \\ &= \sum_{j=2}^N \Delta_j \left(\sum_{t=1}^T \mathbf{P}\{I_t = j\} \right). \end{aligned} \quad (13)$$

We now bound $\mathbf{P}\{I_t = j\}$ for each $j \in \{2, \dots, N\}$. To this end, define random variables

$$\hat{X}_{j,t} = \begin{cases} X_{j,t}/Q_{j,t} & \text{if } I_t = j, \\ 0 & \text{otherwise} \end{cases}$$

where the probability $Q_{j,t} = \mathbf{P}\{I_t = j \mid \mathcal{F}_{t-1}\}$ is defined in (5). Let \mathcal{I}_t be the subset of $\{1, \dots, N\}$ such that, for each $i \in \mathcal{I}_t$,

$$\sum_{s=1}^{t-1} \hat{X}_{i,s} = \max_{1 \leq j \leq N} \sum_{s=1}^{t-1} \hat{X}_{j,s}.$$

For any fixed j , we have

$$\begin{aligned}
\mathbf{P}\{I_t = j\} &= \mathbf{P}\{I_t = j \mid j \in \mathcal{I}_t\} \mathbf{P}\{j \in \mathcal{I}_t\} \\
&\quad + \mathbf{P}\{I_t = j \mid j \notin \mathcal{I}_t\} \mathbf{P}\{j \notin \mathcal{I}_t\} \\
&= \left(\frac{1-\gamma_t}{|\mathcal{I}_t|} + \frac{\gamma_t}{N}\right) \mathbf{P}\{j \in \mathcal{I}_t\} + \frac{\gamma_t}{N} \mathbf{P}\{j \notin \mathcal{I}_t\} \\
&= \frac{1-\gamma_t}{|\mathcal{I}_t|} \mathbf{P}\{j \in \mathcal{I}_t\} + \frac{\gamma_t}{N} \\
&\leq \mathbf{P}\{j \in \mathcal{I}_t\} + \frac{\gamma_t}{N} \\
&= \mathbf{P}\left\{\min_{i \neq j} \sum_{s=1}^{t-1} (\hat{X}_{j,s} - \hat{X}_{i,s}) \geq 0\right\} + \frac{\gamma_t}{N} \\
&\leq \mathbf{P}\left\{\sum_{s=1}^{t-1} (\hat{X}_{j,s} - \hat{X}_{1,s}) \geq 0\right\} + \frac{\gamma_t}{N} \\
&= \mathbf{P}\left\{\sum_{s=1}^{t-1} \eta_t \hat{Z}_{j,s} \geq \eta_t(t-1)\Delta_j\right\} + \frac{\gamma_t}{N} \\
&\quad \text{for } \hat{Z}_{j,t} = \hat{X}_{j,t} - \hat{X}_{1,t} + \Delta_j \text{ and } \eta_t > 0 \text{ arbitrary} \\
&= \mathbf{P}\left\{e^{\sum_{s=1}^{t-1} \eta_t \hat{Z}_{j,s} - \eta_t(t-1)\Delta_j} \geq 1\right\} + \frac{\gamma_t}{N} \\
&\leq \mathbf{E}\left[e^{\sum_{s=1}^{t-1} \eta_t \hat{Z}_{j,s} - \eta_t(t-1)\Delta_j}\right] + \frac{\gamma_t}{N} \\
&\quad \text{as } \mathbf{P}\{X \geq 1\} \leq \mathbf{E}[X] \text{ for any positive r.v. } X \\
&= e^{-\eta_t(t-1)\Delta_j} \mathbf{E}\left[\prod_{s=1}^{t-1} \mathbf{E}\left[e^{\eta_t \hat{Z}_{j,s}} \mid \mathcal{F}_{s-1}\right]\right] + \frac{\gamma_t}{N}. \quad (15)
\end{aligned}$$

The proof is concluded by noting that γ_t is chosen as in (2), η_t can be set as in (3), and (15) is thus equal to (9) as in the proof of Theorem 3.1. \square

Proof of Theorem 3.3. Fix a positive integer T and choose any integer $r \geq 0$. Let $\hat{\mu}_{j,r} = s_{j,r}/T_r$, where $s_{j,r}$ is the total reward for arm j during round r . By hypothesis, $\mathbf{E}[X_{j,t} \mid \mathcal{F}_{t-1}] = \mu_j$ for all $j = 1, \dots, N$ and all t . Thus we can apply Hoeffding bounds [8] and obtain,² for each fixed j and for $\Delta = \Delta(\mu_1, \dots, \mu_N)$,

$$\begin{aligned}
\mathbf{P}\{|\hat{\mu}_{j,r} - \mu_j| > \Delta/2\} &\leq 2e^{-\Delta^2 T_r/2} \\
&\leq 2e^{-d^2 T_r/2} \leq \frac{1}{N2^r}.
\end{aligned}$$

Hence, $\mathbf{P}\{\exists j \mid \hat{\mu}_{j,r} - \mu_j| > \Delta/2\} \leq 2^{-r}$. Therefore, the regret during round r is at most

$$\left(\sum_j \Delta_j\right) T_r + 2^r \frac{1}{2^r} \leq D \lceil 2(\log(2N) + r)/d^2 \rceil + 1.$$

²Note that Hoeffding bounds can be applied, without modification, also to the more general bandit model where the reward distributions can adversarially change after each play provided the reward expectations are kept fixed.

Let ℓ the total number of rounds. That is, ℓ is the smallest integer such that

$$\sum_{r=0}^{\ell} (NT_r + 2^r) \geq T.$$

Clearly, $\ell \leq \ell'$, where ℓ' is the smallest integer such that $\sum_{r=0}^{\ell'} 2^r \geq T$. Thus $\ell + 1 \leq \lceil \log(T + 1) \rceil$. Without loss of generality, assume the last round ends exactly at time T (if it ends before, then the bound gets better). We find that

$$\begin{aligned}
&\max_{1 \leq j \leq N} \mathbf{E}\left[\sum_{t=1}^T (X_{j,t} - X_{I_t,t})\right] \\
&\leq D \sum_{r=0}^{\ell} \lceil 2(\log(2N) + r)/d^2 \rceil + \ell + 1 \\
&\leq D \left(\frac{2\log(2N)}{d^2}(\ell + 1) + \frac{\ell(\ell + 1)}{2d^2}\right) + \ell + 1 \\
&\leq \left(\frac{2D\log(2N)}{d^2} + 1\right)(\ell + 1) + \frac{D}{2d^2}(\ell + 1)^2 \\
&\leq \left(\frac{2D\log(2N)}{d^2} + 1\right) \lceil \log(T + 1) \rceil \\
&\quad + \frac{D}{2d^2} \lceil \log(T + 1) \rceil^2.
\end{aligned}$$

This concludes the proof. \square

Proof of Corollary 3.4. Following (13), the regret of GREEDYMIX can be written as

$$\begin{aligned}
&\max_{1 \leq j \leq N} \mathbf{E}\left[\sum_{t=1}^T (X_{j,t} - X_{I_t,t})\right] \\
&= \sum_{j=2}^N \Delta_j \left(\sum_{t=1}^T \mathbf{P}\{I_t = j\}\right) \\
&\leq \sum_{t=1}^T \sum_{\{i: \mu_i > \mu_1 - d\}} \Delta_i \mathbf{P}\{I_t = i\} \\
&\quad + \sum_{t=1}^T \sum_{\{j: \mu_j \leq \mu_1 - d\}} \Delta_j \mathbf{P}\{I_t = j\} \\
&\leq \left(\max_{\{i: \mu_i > \mu_1 - d\}} \Delta_i\right) T \\
&\quad + \sum_{t=1}^T \sum_{\{j: \mu_j \leq \mu_1 - d\}} \Delta_j \mathbf{P}\{I_t = j\} \\
&= \Delta_{i(d)} T + \sum_{t=1}^T \sum_{\{j: \mu_j \leq \mu_1 - d\}} \Delta_j \mathbf{P}\{I_t = j\}.
\end{aligned}$$

The proof now goes along the same lines as the proof of Theorem 3.1. The analysis of the regret of SOFTMIX is similar. \square

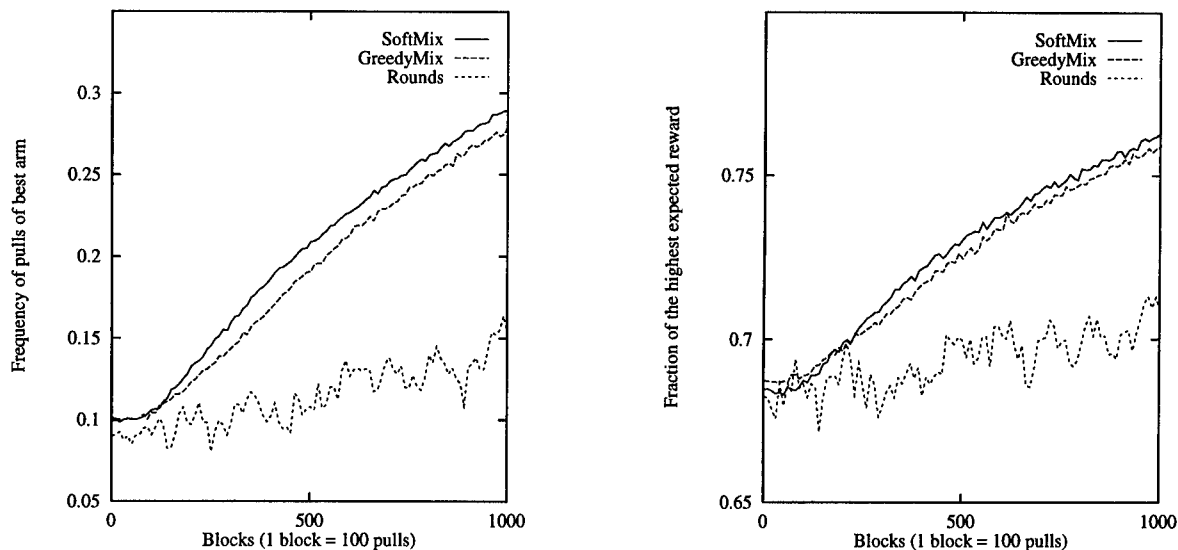


Figure 4: The two graphs are averages over 1,000 runs of 100,000 pulls each. We divided each run in 1000 blocks of 100 pulls each. At **left** we plot the fraction of times the true best arm was pulled in each block. At **right** we plot the average reward per block divided by the highest expected reward for that run.

6 EXPERIMENTS

We tested the three algorithms on a ten-armed bandit problem. Due to the boundedness condition, the rewards were drawn from beta distributions whose range is the unit interval $[0, 1]$. In our experiments we averaged 1,000 runs. In each run the two parameters of the beta distributions were chosen uniformly and independently for each arm from the real interval $[2, 12]$. The parameter d was set to the best possible choice $\Delta(\mu_1, \dots, \mu_N)$. All other constants were set as shown in Figures 1, 2, and 3. In the plots of Figure 4 we compare the performance of SOFTMIX, GREEDYMIX, and ROUNDS on 100,000 pulls. Observe that SOFTMIX and GREEDYMIX have a similar performance, although SOFTMIX performs slightly better after a slower short initial phase. ROUNDS has the slowest convergence, probably due to the long initial domination of explorative phases. Tests with up to 400,000 pulls show that the ranking of the three algorithms stays the same, though ROUNDS considerably improves in the long run as exploitation takes over exploration. Note that our setting of the constants for the mixing coefficient γ_t is independent of any property of the reward distributions other than the parameter d . Hence, it is conceivable (as we indeed observed in the experiments) that more informed choices of the constants in γ_t could lead to a better empirical performance for specific reward distributions. Finally, we ran tests for two other distributions of the rewards: Bernoulli distribution (rewards in $\{0, 1\}$ with expectation of reward 1 chosen independently and uniformly from $[0, 1]$ for every arm) and uniform distributions on $[0, a]$, where the parameter a is chosen independently and uniformly from $[0, 1]$ for every arm. The empirical results did not differ much from those obtained for the beta

distribution. We plan to carry out experiments in order to test our algorithms against Boltzmann Exploration and Interval Estimation.

7 CONCLUSIONS

The main contribution of this paper is the derivation of finite-time regret bounds for variants of widely used heuristics for the bandit problem. Our results demonstrate that the average reward per pull obtained by any one of these variants converges to that of the best arm, and we show explicit bounds on the convergence rate. We remark that, rather than improving the empirical performance on specific domains, our main interest is the understanding of the nature of basic methods like Boltzmann Exploration, and the derivation of rigorous regret bounds that are guaranteed to hold in a vast range of situations.

Our work can be extended in many ways. A more general version of the bandit problem is obtained by removing the stationarity assumption on reward expectations (see [4, 6] for extensions of the basic bandit problem). For example, suppose that a stochastic reward process $\{X_i(s) : s = 1, 2, \dots\}$ is associated to each arm $i = 1, \dots, N$. Here, pulling arm i at time t yields a reward $X_i(s)$ and causes the current state s of arm i to change to $s + 1$, whereas the states of the other arms remain frozen. A well studied problem in this setup is the maximization of the total expected reward in a sequence of T pulls. There are methods, like the Gittins allocation indices, that allow to find the optimal arm to pull at each time t by considering each reward process independently from the others (even though the globally optimal solution depends on all the pro-

cesses).³ However, computation of the Gittins indices requires preliminary knowledge about the reward processes. To overcome this requirement, one can learn the Gittins indices, as proposed in [5] for the case of finite-state Markovian reward processes. However, there are no finite-time regret bounds shown for this solution. At the moment, we do not know whether our techniques could be extended to these more general bandit problems.

Another open problem is whether the bounds we prove are tight for each one of the three algorithms, and whether they are optimal for the bandit problem considered here.

Acknowledgements

Both authors gratefully acknowledge support from ES-PRIT Working Group EP 27150, Neural and Computational Learning II (NeuroCOLT II). Thanks to the referees for their many useful suggestions.

References

- [1] R. Agrawal. Sample mean based index policies with $O(\log n)$ regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27:1054–1078, 1995.
- [2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pages 322–331. IEEE press, 1995.
- [3] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.
- [4] D.A. Berry and B. Fristedt. *Bandit Problems*. Chapman and Hall, 1985.
- [5] M.O. Duff. Q-learning for bandit problems. In *Proceedings of the 12th International Conference on Machine Learning*, pages 209–217. Morgan Kaufmann, 1995.
- [6] J.C. Gittins. *Multi-Armed Bandit Allocation Indices*. Wiley-Interscience series in Systems and Optimization. John Wiley and Sons, 1989.
- [7] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1985.
- [8] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
- [9] T. Ishikida and P. Varaiya. Multi-armed bandit problem revisited. *Journal of Optimization Theory and Applications*, 83(1):113–154, 1994.
- [10] L.P. Kaelbling. *Learning in Embedded Systems*. MIT Press, Cambridge, 1993.
- [11] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [12] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [13] S.R. Kulkarni and G. Lugosi. Mimimax lower bounds for the two-armed bandit problem. Technical Report, Pompeu Fabra University, Barcelona, Spain, 1998.
- [14] T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
- [15] J. Neveu. *Discrete Parameter Martingales*. North-Holland, 1975.
- [16] R.S. Sutton and A.G. Barto. *Reinforcement Learning, an Introduction*. MIT Press / Bradford Books, Cambridge, 1998.
- [17] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
- [18] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

³See [9] for an application of the Gittins indices to the average (undiscounted) reward criterion used here.

Bayesian Classifiers are Large Margin Hyperplanes in a Hilbert Space

Nello Cristianini

Dept of Engineering Maths
University of Bristol,
Bristol, UK

nello.cristianini@bristol.ac.uk

John Shawe-Taylor

Dept of Computer Science
RHBNC
Egham, UK

jst@dcs.rhbnc.ac.uk

Peter Sykacek

Austrian Research Institute
for Artificial Intelligence
Vienna, Austria

peter@ai.univie.ac.at

Abstract

Bayesian algorithms for Neural Networks are known to produce classifiers which are very resistant to overfitting. It is often claimed that one of the main distinctive features of Bayesian Learning Algorithms is that they don't simply output one hypothesis, but rather an entire distribution of probability over an hypothesis set: the Bayes posterior. An alternative perspective is that they output a linear combination of classifiers, whose coefficients are given by Bayes theorem. One of the concepts used to deal with thresholded convex combinations is the 'margin' of the hyperplane with respect to the training sample, which is correlated to the predictive power of the hypothesis itself.

We provide a novel theoretical analysis of such classifiers, based on Data-Dependent VC theory, proving that they can be expected to be large margin hyperplanes in a Hilbert space. We then present experimental evidence that the predictions of our model are correct, i.e. that bayesian classifiers really find hypotheses which have large margin on the training examples.

This not only explains the remarkable resistance to overfitting exhibited by such classifiers, but also collocates them in the same class of other systems, like Support Vector machines and Adaboost, which have a similar performance.

Keywords: Bayesian Classifiers, Large margin hyperplanes, Hilbert space

1 INTRODUCTION

Bayesian learning algorithms for neural networks of the kind described in [3] are often claimed to have the distinctive feature of outputting an entire distribution of probability over the hypothesis space, rather than a single hypothesis. Such a distribution, the Bayes

posterior, depends on the training data and on prior distribution, and is used to make predictions by averaging the predictions of all the elements of the set, in a weighted majority voting scheme.

The posterior is computed according to Bayes' rule, and such a scheme has the remarkable property that - as long as the prior is correct and the computations can be performed exactly - its expected test error is minimal. Typically, the posterior is approximated by combining a gaussian prior and a simplified version of the likelihood (the data-dependent term, that is the term that reflects the information gleaned from the training set). Such a distribution is then sampled with a Montecarlo method, to form a committee whose composition reflects the posterior probability. The predictive integral over a posterior distribution can hence be replaced by a sum.

The classifiers obtained with this method are known to be highly resistant to overfitting. Indeed, neither the committee size nor the network size strongly affect the performance, to such an extent that it is not uncommon - in the bayesian literature - to find computations with "infinite networks" [4], [10], meaning by this the posterior over the complete (infinite) hypothesis space. Statistical Learning Theory, on the other hand, is concerned with the problem of bounding the test error (in the worst case and with high probability) using quantities that are observable in the training set or known a priori [9].

The expressions obtained for such a bound typically depend on the training error, the sample size and the VC dimension of the classifier. Given that the number of tunable parameters gives a rough estimation of the VC dimension, the size of the network and that of the committee do matter.

A more refined, Data-Dependent, version of the theory introduced in [8], shows that it is possible to replace

the VC dimension in the above mentioned bounds with a quantity which depends on the margin of the classifier on the training examples.

In this paper we provide a novel description of Bayesian classifiers which makes it possible to perform margin analysis on them, and hence to apply Data-Dependent VC theory. In particular, by viewing the posterior distribution as a linear functional in a Hilbert space, the margin can be computed and gives a bound on the generalization error via an 'effective' VC dimension which is much lower than the number of parameters.

Finally, experimental study is performed with a standard bayesian algorithm [5] on real world data, in order to test the predictions of our model. The results of the experiments confirm that the model captures the relevant features of these classifiers, and that they can indeed be regarded as large margin hyperplanes in a Hilbert space.

Margin-distribution graphs are provided for different data sets, different network sizes, committee sizes and choices of prior, always showing the same qualitative behaviour: a clear bias toward large margin on training examples.

Our plots can be directly compared with the ones presented in the inspiring paper by Shapire et al. [7], where this concept was introduced, as we have used the same datasets. In that paper, a bound on the test error as a function of the margin distribution was first obtained.

These theoretical and experimental results not only explain the remarkable resistance to overfitting observed in bayesian algorithms, but also provide a surprising unified description of three of the most effective learning algorithms: Support Vector Machines, Adaboost and now also Bayesian classifiers.

2 BAYESIAN LEARNING THEORY

The result of Bayesian learning is a probability distribution over the (parametrized) hypothesis space, expressing the degree of belief in a specific hypothesis as approximation of the target function. Such distribution is then used to make predictions.

To start the process of bayesian learning, one must define a prior distribution $P(w)$ over the parameter space, possibly encoding some prior knowledge. After observing the data, the prior distribution is updated using Bayes' Rule:

$$P(w|D) \propto P(D|w)P(w),$$

where $P(w|D)$ is the probability of the parameters given the data D , $P(D|w)$ the probability of the data given the parameters, and $P(w)$ the prior distribution over the parameters. The posterior distribution so obtained, hence, encodes information coming from the training set (via the likelihood function $P(D|w)$) and prior knowledge.

To predict the label of a new point, bayesian classifiers integrate the predictions made by every element of the hypothesis space, weighting them with the posterior associated to each hypothesis, obtaining a distribution of probability over the set of possible labels (note that h_w is the function parametrised by w):

$$P(y|x, D) = \int_w h_w(x)p(w|D)dw$$

This predictive distribution can be used to minimize the number of misclassifications in the test set; in the 2-class case this is achieved simply by outputting the label which has received the highest vote.

3 BAYESIAN CLASSIFIERS AS LARGE MARGIN HYPERPLANES

Hence, the actual hypothesis space used by Bayesian systems is the Convex Hull of H , rather than H . The output hypothesis is a hyperplane, whose coordinates are given by the posterior.

In order to study the margin of such hyperplanes, we will introduce some simplifications in the general model. We assume that the base hypothesis space, H is formed by Boolean valued functions, and that it is sufficiently rich that all dichotomies can be implemented. Further, initially we will assume that the average prior probability over functions in a particular error shell does not depend on the number of errors.

These are the only assumptions we make, and the second will to be relaxed in a second stage. A natural choice for the evidence function in a Boolean valued hypothesis space is $e^{-k\sigma}$, where k is the number of mistakes made by the hypothesis and $\sigma > 0$ an appropriately chosen constant. The expression has the required property of giving low likelihood to the predictors which make many mistakes on the training set, and to which the usual Bayesian evidence collapses in the Boolean case. Our analysis will also suggest suitable choices for σ .

It can be interpreted with an assumption of Gaussian noise corrupting the data after they have been labelled by a target function which belongs to H , the variance of the noise depending on $1/\sigma$.

The assumption that all the dichotomies can be implemented with the same probability corresponds to an 'uninformative' prior, where no knowledge is available about the target function. In a second stage we will examine the effect of inserting some knowledge in the prior, by slightly perturbing the uninformative one towards the target hypothesis. We will see that even slightly favourable priors can give a much smaller VC dimension than the uninformative one.

3.1 THE UNINFORMATIVE PRIOR

The actual hypothesis space used by Bayesian systems, hence, is the Convex Hull of H , rather than H . The output hypothesis is a hyperplane, whose coordinates are given by the posterior.

In this section we give an expression for the margin of the composite hypothesis, as a function of a parameter related to our model of likelihood. The result is obtained in the case of a uniform prior, and for the pattern recognition case.

Let us start by stating some simple results and definitions which will be useful in the following.

Definition 3.1 Let B_i be the balance of the hypothesis h_i over a given sample of size m , that is the number of successes s_i minus the number of failures f_i : $B_i = s_i - f_i$, $m = s_i + f_i$.

Therefore $B_i = m - 2f_i$, which implies $B_i/m = 1 - 2\epsilon_i$, where $\epsilon_i = f_i/m$ is the empirical error of h_i .

During the next proof we will need to know the probability in the prior distribution of hypotheses in our parameter space with a fixed empirical error. Given that this information is in general not available, we will initially make the simplifying assumption that all behaviours on the training sample can be realised. This implies that the hypothesis space has VC dimension greater than or equal to the sample size m .

We make the further assumption that the prior probability of hypotheses which have error $\epsilon = k/m$ is

$$\frac{1}{2^m} \binom{m}{k} = \frac{m!}{2^m (m\epsilon)! (m - m\epsilon)!},$$

in other words that the average prior probability for functions realising different patterns of k errors is 2^{-m} . We will assume that the posterior distribution for a hypothesis which has k training errors is proportional to $e^{-\sigma k} = C^k$, where $C = e^{-\sigma}$. We are now ready to give the main result of this section.

Theorem 3.2 Under the above assumptions the mar-

gin of the Bayes Classifier is given by

$$1 - \frac{2C}{1+C}.$$

Proof: Let the set of training examples be (x_1, \dots, x_m) with classifications $\mathbf{y} = (y_1, \dots, y_m) \in \{-1, 1\}^m$. Let the margin M of example i be M_i . Consider first the average margin

$$\begin{aligned} \langle M \rangle &= \frac{1}{m} \sum_{i \in S} M_i = \frac{1}{m} \sum_{i \in S} y_i F(x_i) \\ &= \frac{1}{m} \sum_{i \in S} y_i \int_H a_h h(x_i) dP(h) \\ &= \frac{1}{m} \sum_{i \in S} y_i \sum_{j \in J} a_j P_j h_j(x_i) \end{aligned}$$

where h_j , $j \in J$ are representatives of each possible classification of the sample. We are denoting by P_j the prior probability of classifiers agreeing with h_j . The quantity $a_j P_j$ is the posterior probability of these classifiers, where the coefficient $a_j = A e^{-\sigma m \epsilon_j} = A C^{m \epsilon_j}$ is the evidence, which depends only on the empirical error and the normalising constant A . By assumption, we have

$$\sum_{k \text{ error shell}} P_j = \binom{m}{k} \frac{1}{2^m}.$$

Hence,

$$\begin{aligned} \langle M \rangle &= \frac{1}{m} \sum_{j \in J} a_j P_j \sum_{i \in S} y_i h_j(x_i) \\ &= \frac{1}{m} \sum_{j \in J} a_j P_j B_j \\ &= \sum_{j \in J} a_j P_j (1 - 2\epsilon_j) \\ &= 1 - 2 \sum_{j \in J} a_j P_j \epsilon_j \end{aligned} \quad (1)$$

by the observation concerning the balance B_j of h_j and the fact that the posterior distribution has been normalised, that is $1 = \int_H a_h dP(h) = \sum_{j \in J} a_j P_j$.

We now regroup the elements of the sum on the right hand side of the above equation by decomposing the hypothesis space into error shells. Hence, we can write the above sum as

$$\sum_{j \in J} a_j P_j \epsilon_j = \frac{1}{2^m} \sum_{k=0}^m A C^k \binom{m}{k} \frac{k}{m}. \quad (2)$$

Solving for A and substituting, gives

$$\sum_{j \in J} a_j P_j \epsilon_j = \frac{\sum_k C^k \binom{m}{k} \frac{k}{m}}{\sum_k C^k \binom{m}{k}}$$

We can now use the equality $\sum_k C^k \binom{m}{k} = (1 + C)^m$, and the observation that $\sum_k C^k \binom{m}{k} k$ can be written as $C \frac{d}{dC} \sum_k C^k \binom{m}{k} = mC(1 + C)^{m-1}$ to obtain the result for the average margin.

To complete the proof we must show that the average margin is in fact the minimal margin. We will demonstrate this by showing that the margin of all points is equal. Intuitively, this follows from the symmetry of the situation, there being nothing to distinguish between different training points in the structure of the hypothesis. The formal proof relies on performing a permutation on the training points, but has had to be omitted in this shortened version. ■

There are three relevant bounds on the generalization error in terms of the margin on the training set. We will quote all three here and then discuss their applicability in the current context. The first two appear in Schapire *et al.* [7].

Following [7], let H denote the space from which the base hypotheses are chosen (for example Neural Networks, or Decision Trees). A base hypothesis $h \in H$ is a mapping from an instance space X to $\{-1, +1\}$.

Theorem 3.3 *Let S be a sample of m examples chosen independently at random according to D . Assume that the base hypothesis space H has VC dimension d , and let be $\delta > 0$. Then, with probability at least $1 - \delta$ over the random choice of the training set S , every weighted average function $f \in C$ satisfies the following bound for all $\theta > 0$:*

$$P_D[yF(x) \leq 0] \leq P_S[yF(x) \leq \theta] + O\left(\frac{1}{\sqrt{m}} \left(\frac{d \log^2(m/d)}{\theta^2} + \log(1/\delta)\right)^{1/2}\right)$$

Theorem 3.4 *Let S be a sample of m examples chosen independently at random according to D . Assume that the base hypothesis space H is finite, and let be $\delta > 0$. Then, with probability at least $1 - \delta$ over the random choice of the training set S , every weighted average function $f \in C$ satisfies the following bound for all $\theta > 0$:*

$$P_D[yF(x) \leq 0] \leq P_S[yF(x) \leq \theta] + O\left(\frac{1}{\sqrt{m}} \left(\frac{\log^2(m) \log |H|}{\theta^2} + \log(1/\delta)\right)^{1/2}\right)$$

As observed by the authors, the theorem applies to every majority vote method, including boosting, bagging, ECOC, etc.

The third is contained in Shawe-Taylor *et al.* [8] and involves the fat shattering dimension of the space of functions.

Theorem 3.5 *Consider a real valued function class \mathcal{F} having fat shattering function bounded above by the function $\text{afat} : \mathbb{R} \rightarrow \mathbb{N}$ which is continuous from the right. Fix $\theta \in \mathbb{R}$. If a learner correctly classifies m independently generated examples \mathbf{z} with $h = T_\theta(f) \in T_\theta(\mathcal{F})$ such that $\text{er}_{\mathbf{z}}(h) = 0$ and $\gamma = \min |f(x_i) - \theta|$, then with confidence $1 - \delta$ the expected error of h is bounded from above by*

$$\epsilon(m, k, \delta) = \frac{2}{m} \left(k \log \left(\frac{8em}{k} \right) \log(32m) + \log \left(\frac{8m}{\delta} \right) \right),$$

where $k = \text{afat}(\gamma/8)$.

Since the assumption that the underlying hypothesis space can perform any classification of the training set implies that its VC dimension is at least m , we cannot expect that learning is possible in the situation described. Indeed, we have augmented the power of the hypothesis space by taking our functions from the convex hull of H which would appear to make the situation yet worse.

Hence, in order to obtain useful applications of any of the theorems we will need to consider deviations from the most general situation described above. The deviation should not have a significant impact on the margin, while reducing the expressive power of the hypotheses.

In order to apply Theorem 3.4 the number of hypotheses in the base class H must be finite. The logarithm of the number of hypotheses appears in the result. Since we have assumed that all possible classifications of the training set can be performed the number of hypotheses must be at least 2^m making the bound uninteresting. To apply this theorem we must assume that a very large proportion of the hypotheses have zero weight in the prior, while those that have significant weights in the posterior (i.e. have low empirical error) are retained. Making this assumption the bound will become significant. However, we are interested in capturing the effect of non-discrete priors, that is situations where potentially all of the base hypotheses are included, but those with high empirical error have lower prior probability.

In order to apply Theorem 3.3 the underlying hypothesis class H must be assumed to have low VC dimension

in such a way that no significant impact is made on the margin. This could be achieved by removing high error functions. Note that the functions would have to be removed, in other words given prior probability 0. Hence, the bound obtained would be no better than a standard VC bound in the original space. A situation where this approach and analysis might be advantageous is where the consistent hypothesis $h_{\mathbf{y}}$ is not included in H . This will reduce the margin by approximately $a_{h_{\mathbf{y}}} 2^{-m} = (1+C)^{-m}$, since $B_{h_{\mathbf{y}}} = m$ (see equation (1)). The approximation arises from not adjusting the normalisation to take account of the missing hypothesis and is thus a very small error.

These applications are unable to take into account the prior distribution in a flexible way. In the next section we will present an application of the third approach to show how this can take advantage of a beneficial prior.

3.2 THE EFFECT OF THE PRIOR DISTRIBUTION ON THE MARGIN BOUND

We will consider the situation where the prior decays arithmetically with the error shells. In other words the prior on hypotheses with error k is multiplied by α^k for some $\alpha < 1$. We first repeat the calculations of Theorem 3.2 for this case. The sum (2) must take into account that in this case

$$\sum_{k \text{ error shell}} P_j = \alpha^k (1 + \alpha)^{-m} \binom{m}{k}.$$

The factor $(1 + \alpha)^m$ cancels and the factor α appears wherever C appears, that is

$$\sum_{j \in J} a_j P_j \epsilon_j = \frac{1}{(1 + \alpha)^m} \sum_{k=0}^m A C^k \alpha^k \binom{m}{k} \frac{k}{m},$$

while

$$\frac{A}{(1 + \alpha)^m} \sum_{k=0}^m C^k \alpha^k \binom{m}{k} = 1.$$

Hence, the margin can be computed as

$$1 - \frac{2\alpha C}{1 + \alpha C}.$$

We now quote a theorem due to Gurvits [2] that bounds the fat shattering dimension of linear functionals in Banach spaces which we will need to bound the effective VC dimension.

Theorem 3.6 [2] *Consider a Banach space B of type p and the class of linear functions L of norm less than or equal to one restricted to the unit sphere. Then there is a constant D such that $\text{fat}_L(\gamma) \leq D\gamma^{-p/(p-1)}$.*

Note that for Hilbert spaces which we will consider the value of $p = 2$.

In order to apply Theorems 3.5 and 3.6 we need to bound the radius of the sphere containing the points and the norm of the linear functionals involved. Clearly, scaling by these quantities will give the margin appropriate for application of the theorem. The Hilbert space we consider is that given by the input space X with inner product

$$\langle x, y \rangle = \int_H h(x)h(y)dP(h).$$

Hence, the norm of input points is 1 and they are contained in the unit sphere as required. The linear functionals considered are those determined by the posterior distribution. The norm is given by

$$\|a\|^2 = \int_H a_h^2 dP(h).$$

We must compute this value for the posterior functional in the prior described above. The integral in this case is given by

$$\begin{aligned} \|a\|^2 &= \sum_{j \in J} a_j^2 P_j = \sum_{k=0}^m A^2 C^{2k} \frac{\alpha^k}{(1 + \alpha)^m} \binom{m}{k} \\ &= \frac{(1 + \alpha)^m (1 + \alpha C^2)^m}{(1 + \alpha C)^{2m}}. \end{aligned}$$

Hence, the bound on the fat shattering dimension becomes,

$$g(\alpha, C) := \frac{(1 + \alpha)^m (1 + \alpha C^2)^m}{(1 + \alpha C)^{2m-2} (1 - \alpha C)^2}.$$

In the rest of this section we will consider how this function behaves for various choices of C and α , showing that for careful choices of C , values of α close to 1 can give dimensions significantly lower than m , hence give good bounds on the generalization error. The analysis shows that using this approach it is possible to make use of a beneficial prior. At the same time it suggests a value of C most likely to take advantage of such a prior.

First consider the case when $\alpha = 1$. Hence,

$$g(1, C) = \frac{2^m (1 + C^2)^m}{(1 + C)^{2m-2} (1 - C)^2}.$$

The parameter C can be chosen in the range $[0, 1)$. However, $g(1, C) \rightarrow_{C \rightarrow 1} \infty$, while $g(1, 0) = 2^m$. Clearly, the optimal choice of C needs to be determined if the bound is to be useful. A routine calculation establishes that the value of C which minimises

the expression is, $C_0 = (m - \sqrt{m-1})/(m-2)$, which gives a value of

$$g(1, C_0) = m \left(1 + \frac{1}{m-1}\right)^{m-1} \approx em.$$

This confirms that the effective VC dimension is not increased excessively provided C is chosen around $1 - 2/\sqrt{m}$. In order to study the effect of allowing α to move slightly below 1, we will perform a Taylor expansion about $\alpha = 1$.

Let $C' = \alpha C$ and the function

$$g_1(\alpha, C') := g(\alpha, C'/\alpha) = \frac{(1+\alpha)^m (1+C'^2/\alpha)^m}{(1+C')^{2m-2} (1-C')^2}.$$

Note that $\frac{\partial g_1(\alpha, C')}{\partial C'} \Big|_{\alpha=1} = 0$, and so $\frac{\partial g(\alpha, C_0)}{\partial \alpha} =$

$$\frac{\partial g_1(\alpha, C')}{\partial \alpha} + \frac{\partial g_1(\alpha, C')}{\partial C'} \frac{dC'}{d\alpha}. \text{ Hence,}$$

$$\frac{\partial g(\alpha, C_0)}{\partial \alpha} \Big|_{\alpha=1} = \frac{\partial g_1(\alpha, C')}{\partial \alpha} \Big|_{\alpha=1}.$$

Differentiating gives

$$\frac{\partial g_1(\alpha, C')}{\partial \alpha} \Big|_{\alpha=1} = \frac{m2^{m-1}(1+C'^2)^{m-1}}{(1+C')^{2m-3}(1-C')}$$

We can now perform a Taylor series expansion of $g(\alpha, C_0)$ about $\alpha = 1$ to obtain $g(\alpha, C_0) \approx em(1 + (\alpha - 1)\sqrt{m-1})$, where we have omitted some routine calculations. Hence, the bound on the generalization error is (ignoring log factors) $\tilde{O}(1 - (1 - \alpha)\sqrt{m-1})$, so that to obtain generalization error of order ϵ , we need

$$\alpha \approx 1 - \frac{1 - \epsilon}{\sqrt{m-1}}.$$

Hence, for values of α very close to 1, the prior can result in very good generalization properties.

4 EXPERIMENTS

In this section we will look at some experiments where we calculated margin distributions for two data sets. We used the vehicle data and the satimage data, both taken from the StatLog¹ database. These datasets were used by [7] for a comparison of the margin distributions of Bagging and Boosting. We used satimage as provided, there are 4435 samples in the training and 2000 in the test set. The vehicle data were merged, 500 samples were used for training and 252 for testing.

¹The data are available via the UCI machine learning repository at <http://www.ics.uci.edu/mllearn/MLRepository.html>.

4.1 EXPERIMENTAL SETUP

Both datasets are polychotomous classification problems. To arrive at a reasonable posterior probability density over weight space besides a prior we need a proper data model and likelihood term.

According to [1], the best thing we can do in the case of polychotomous classification is to use (3), the generalized logistic or softmax transformation of the output layer activations. Given distributions of hidden unit activations, which are members of the exponential family, this transformation guarantees that the network outputs may be interpreted as probabilities for classes.

$$p(C_k | \mathbf{z}) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})} \quad (3)$$

In (3) the value a_k is the value at output node k before applying softmax activation.

Having sampled a sufficient number of weights we are ready to predict. In a Bayesian framework each input value leads to a predictive distribution of network outputs. In the case of classifications, the network output is simply given by integrating over the predictive distribution. Having sampled from the posterior over weights, in our case the expectation is approximated by a sum over the weights.

The experiments were performed for both datasets with different settings. Initially we sampled 600 weights using the standard method without ARD-priors (Automatic Relevance Determination [3]). The network size was fixed to 25 hidden units for both datasets. This experiment was used to investigate the dependence of the margin distribution of the number of weights used to represent the posterior. Discarding 50 initial weights, we calculated the margin distribution of a committee consisting of the next 150 weights and compared it to the margin distribution when using all 550 remaining weights.

To assess whether the margin distribution changes while increasing the size of the network, we performed two further experiments sampling 150 weights for a network with 50 and 200 hidden units respectively, again using conventional priors without ARD. A fourth experiment should reveal the influence of an ARD-prior on the margin distribution. We sampled 150 weights for a network with 25 hidden units using an ARD-prior on the input to hidden layer weights.

Figure 1 shows plots of the resulting margin distributions for the vehicle dataset. The margin distributions for the satimage data are shown in Figure 2. Looking at the plots of the margin distributions, we see that they are different. It is interesting to investigate

whether these differences are significant and whether the differences in the margin distributions are correlated with the performance of the classifier on an independent test set. From theory we expect that a classifier which shows larger margins on the training data should also show a better generalization error.

For both experiments with the 200 hidden units networks we see a trend towards lower margins. This fact can be understood when remembering that the prior variance of the hidden to output weights scales inversely with the number of hidden units. Increasing the number of hidden units forces smaller hidden to output weights which leads to a smaller complexity of the network and therefore to underfitting and increased errors on the training set.

4.2 RESULTS

In order to compare the margin distribution with the generalization error, we used each classifier to predict class labels on an independent test set. The different experimental setups and the resulting generalization errors are summarized in table 1.

Table 1: Network size, information about prior distribution, committee size, and generalization error for satimage (sat) and vehicle (veh) data.

Net size	ARD	Prior	Comm. size	Error sat	Error veh
25	no	$\Gamma(0.05, 0.5)$	150	9.2%	15.5%
25	no	$\Gamma(0.05, 0.5)$	550	8.9%	14.7%
50	no	$\Gamma(0.05, 0.5)$	150	8.6%	13.5%
200	no	$\Gamma(0.05, 0.5)$	150	7.7%	24.2%
25	yes	$\Gamma(0.05, 0.5)$	150	9.7%	17.5%

In order to test our hypothesis that a better performance on the test set is indicated by larger margins on the training data, we will use the first experiment as reference and compare its margin distribution with the margin distributions of the second to fifth experiment.

Four one sided t-tests were used to assess whether the observed differences of means are significant. Assuming independent individual experiments, this approach suffers from the fact that the risk of having incorrectly rejected one of the hypothesis is as large as the sum of the individual significance levels. In this case we get no problem because each experiment was highly significant. In table 2 we show the generalization error, the means of the margin distributions. We expect that

Table 2: Generalization error and margin distributions

Satimage data		Vehicle data	
Error	Mean margin	Error	Mean margin
9.2%	0.929	15.5%	0.73
8.9%	0.932	14.7%	0.72
8.6%	0.926	13.5%	0.78
7.7%	0.898	24.2%	0.45
9.7%	0.895	17.5%	0.70

larger mean values of the margin distribution correspond to smaller generalization errors. Looking at the satimage experiments, we see that this is true for the large committee experiment and for the ARD-prior experiment when compared to the first experiment. For the vehicle data we see the expected correlation for both large network scenarios and for the ARD-prior experiment again comparing with the results of the first experiment.

5 CONCLUSIONS

Our theoretical analysis and experimental results show that Bayesian Classifiers of the kind described in [3] can be regarded as large margin hyperplanes in a Hilbert space, and consequently can be analysed with the tools of Data-Dependent VC theory.

The non-linear mapping from the input space to the Hilbert space is given by the initial choice of network architecture, while the coordinates of the hyperplane are given by the Bayes' posterior and hence depend both on the training data and on the chosen prior.

The choice of the prior turns out to be a crucial one, since we have shown how even slightly correctly guessed priors can be translated into a much lower VC dimension of the resulting classifier (and this - coupled with high training accuracy - ensures good generalization). But even with a totally uninformative prior there is at least no harm in using these apparently overcomplex systems.

Experiments performed on real world data confirm the predictions of the model, highlighting a strong bias toward large margins in all experimental conditions and with different data sets. Their correlation with test error has also been studied.

The practical utility of VC bounds, however, does not lie in quantitative predictions of the test error (the price for their universality is often a certain looseness), but rather in providing an analytical expression of the test error which can be used to study the role of the dif-

ferent parameters and design choices on the final performance. Also, via the SRM principle, such bounds provide a theoretically sound indicator of performance. The results obtained in this work can be incorporated in actual learning systems, to provide for example an independent stopping criterion: the VC bound on the error could be calculated during the learning, and the training could be stopped when no significant increase in performance is observed. Also, the other choices like net size, committee size, type of prior, could be performed using as a guideline their effect on the margin.

On the theoretical side, the surprising result of this paper is to co-locate Bayesian Classifiers in the same category of other systems – namely Support Vector Machines and Adaboost – which were motivated by very different considerations but which exhibited very similar behaviours (e.g. with respect to overfitting).

A unified analysis of the three systems is now possible, which can make potentially fruitful comparisons or cross-fertilizations much easier.

Acknowledgements

Nello Cristianini is funded by EPSRC research grant number GR/L28562.

This work was supported by the Australian Research Council, and the ESPRIT Working Group in Neural and Computational Learning (NeuroCOLT Nr. 8556).

P. Sykacek is funded by the European Commission (Biomed-2 project SIESTA, grant BMH4-CT97-2040). The Austrian Research Institute for Artificial Intelligence is funded by the Austrian Federal Ministry of Science and Transport.

The authors would like to express their gratitude to the Isaac Newton Institute for Mathematical Sciences for having been invited for several stays during the Machine Learning and Neural Networks programm, organized at the institute between August and December 1997. This joint work was initiated there.

Thanks also to Chris Williams for useful discussions and to Radford Neal for making his code freely available over the Internet.

References

- [1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.
- [2] Leonid Gurvits, A note on a scale-sensitive dimension of linear bounded functionals in Banach spaces, In *Proceedings of Algorithmic Learning Theory, ALT-97*, and as NECI Technical Report, 1997.
- [3] Radford Neal, *Bayesian Learning in Neural Networks*, Springer Verlag, 1996
- [4] Radford Neal, *Priors for Infinite Networks*, Technical Report CRG-TR-94-1 (Dept. of Computer Science, University of Toronto), <http://www.cs.toronto.edu/~radford/pin.ps.Z>.
- [5] <http://www.cs.toronto.edu/~radford/software-online.html>
- [6] Carl Rasmussen, *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*, PhD Thesis. <http://www.cs.toronto.edu/pub/carl/thesis.ps.gz>
- [7] R. Schapire, Y. Freund, P. Bartlett, W. Sun Lee, Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods, *Proceedings of the International Conference on Machine Learning (ICML'97)*, 1997
- [8] John Shawe-Taylor, Peter L. Bartlett, Robert C. Williamson, Martin Anthony, Structural Risk Minimization over Data-Dependent Hierarchies, NeuroCOLT Technical Report NC-TR-96-053, 1996. (<ftp://ftp.dcs.rhnc.ac.uk/pub/neurocolt/tech.reports>).
- [9] Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995
- [10] Chris Williams, *Computation with Infinite Networks*, in *Advances in Neural Information Processing Systems*, NIPS 96, Morgan Kaufmann, 1997.

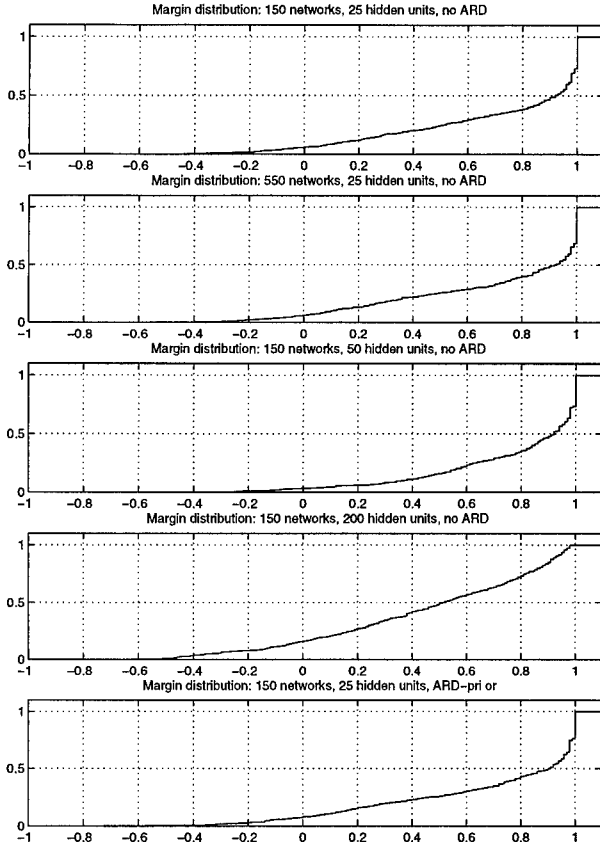


Figure 1: Plot of margin distribution of the vehicle data. The different experimental setups lead to different margin distributions. Further investigations show that these differences are highly significant. Using the first experiment as reference, the third to fifth margin distribution indicate the correct trend in the generalization error for the third to fifth classifier respectively, whereas the conclusion we would draw from the second margin distribution is misleading.

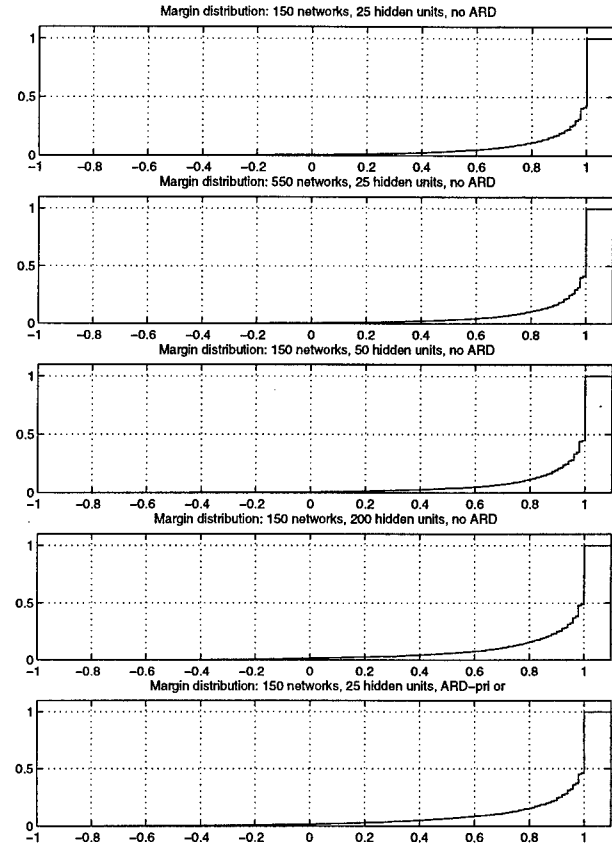


Figure 2: Plot of margin distribution of the satim-age data. Also in this case we get different margin distributions. Again using the first experiment as reference, the margin distributions of these experiments allow to predict the correct trend of the generalization performance for the second and fifth experiment. The conclusion of the third and fourth margin distribution which indicates worse generalization performance compared to the first experiment is again misleading.

The MAXQ Method for Hierarchical Reinforcement Learning

Thomas G. Dietterich
 Department of Computer Science
 Oregon State University
 Corvallis, Oregon 97331
 tgd@cs.orst.edu

Abstract

This paper presents a new approach to hierarchical reinforcement learning based on the MAXQ decomposition of the value function. The MAXQ decomposition has both a procedural semantics—as a subroutine hierarchy—and a declarative semantics—as a representation of the value function of a hierarchical policy. MAXQ unifies and extends previous work on hierarchical reinforcement learning by Singh, Kaelbling, and Dayan and Hinton. Conditions under which the MAXQ decomposition can represent the optimal value function are derived. The paper defines a hierarchical Q learning algorithm, proves its convergence, and shows experimentally that it can learn much faster than ordinary “flat” Q learning. Finally, the paper discusses some interesting issues that arise in hierarchical reinforcement learning including the hierarchical credit assignment problem and non-hierarchical execution of the MAXQ hierarchy.

1 Introduction

Hierarchical approaches to reinforcement learning (RL) problems promise many benefits: (a) improved exploration (because exploration can take “big steps” at high levels of abstraction), (b) learning from fewer trials (because fewer parameters must be learned and because subtasks can ignore irrelevant features of the full state) and (c) faster learning for new problems (because subtasks learned on previous problems can be re-used).

Recent research has explored three general approaches to reaching these goals. The first approach, introduced by Dean and Lin (1995), exploits a hierarchical decomposition primarily as a computational device to accelerate the

computation of the optimal policy. The second approach, introduced by Parr and Russell (1998) relies on a programmer to design a hierarchy of abstract machines that constrains the possible policies to be considered. Their method computes the policy that is optimal subject to these hierarchical constraints by effectively flattening the hierarchy. We will call this kind of policy *hierarchically optimal*, because it is the best policy consistent with the imposed hierarchy. The third approach, pioneered by Singh (1992), Kaelbling (1993), and Dayan and Hinton (1993), also relies on a programmer-designed hierarchy. In this hierarchy, each subtask is defined in terms of goal states or termination conditions. Each subtask in the hierarchy corresponds to its own Markov Decision Problem (MDP), and the methods seek to compute a policy that is locally optimal for each subtask. We will call such policies *recursively optimal*. Recent work by Precup, Sutton, and Singh (1998) studies aspects of both the first third approaches.

In this paper, we extend the research on recursively optimal policies by introducing the MAXQ method for hierarchical reinforcement learning. The methods introduced by Singh, Kaelbling, and Dayan and Hinton are all specific to particular tasks. The Feudal Q learning method of Dayan and Hinton suffers from the problem that at all non-primitive levels of a Feudal-Q hierarchy, the learning task can become non-Markovian, and therefore difficult to solve. In contrast, the MAXQ method is general purpose. At each level of the hierarchy, the task is Markovian and can be solved by standard RL methods. In many cases, state abstractions can be introduced without destroying the optimality of the learned policy. Like Kaelbling’s work, MAXQ supports non-hierarchical execution of the learned policy, which permits it to behave well even when the optimal policy violates the structure of the hierarchy.

This paper is organized as follows. First, we introduce the MAXQ hierarchy using an example and define its procedural and declarative semantics. Then we introduce two theo-

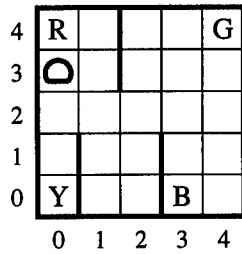


Figure 1: The Taxi Domain

remains that describe the conditions under which the MAXQ hierarchy can successfully represent the value function of a fixed hierarchical policy. Section 4 introduces a learning algorithm for training a MAXQ hierarchy and shows experimentally and theoretically that it works well. Finally, the paper shows how a non-hierarchical policy can be computed and executed using the MAXQ hierarchy.

2 The MAXQ Hierarchy

We will introduce the MAXQ method using the simple Taxi Problem shown in Figure 1. A taxi inhabits a 5-by-5 grid world. There are four specially-designated locations in this world, marked as R(ed), B(lue), G(reen), and Y(ellow). The taxi problem is episodic. In each episode, the taxi starts in a randomly-chosen state and with a randomly-chosen amount of fuel (ranging from 5 to 12 units). There is a passenger at one of the four locations (chosen randomly), and that passenger wishes to be transported to one of the four locations (also chosen randomly). The taxi must go to the passenger's location (the "source"), pick up the passenger, go to the destination location (the "destination"), and put down the passenger there. (To keep things uniform, the taxi must pick up and drop off the passenger even if he/she is already located at the destination!) The episode ends when the passenger is deposited at the destination location.

There are seven primitive actions in this domain: (a) four navigation actions that move the taxi one square North, South, East, or West (each of these consumes one unit of fuel), (b) a Pickup action, (c) a Putdown action, and (d) a Fillup action (which can only be executed when the taxi is at location F(uel)). Each action is deterministic. There is a reward of -1 for each action and an additional reward of $+20$ for successfully delivering the passenger. There is a reward of -10 if the taxi attempts to execute the Putdown or Pickup actions illegally. If a navigation action would cause the taxi to hit a wall, the action is a no-op, and there is only the usual reward of -1 . Finally, the episode also ends (with a reward of -20) if the fuel level falls below

zero.

We seek a policy that maximizes the average reward per step. In this domain, this is equivalent to maximizing the total reward per episode. The optimal policy—which is non-trivial to implement by hand—attains an average reward per step of 0.92 (computed over 5000 trials). There are 8,750 possible states: 25 squares, 5 locations for the passenger (counting the four starting locations and the taxi), 5 destinations, and 14 fuel levels.

This task has a simple hierarchical structure in which there are three sub-tasks: Get the passenger, Refuel the taxi, and Deliver the passenger. Each subtask involves navigating to one of the five locations and then performing a Pickup, Fillup, or Putdown action. While the taxi is navigating to a location, only that location is relevant. We would like to capture this hierarchical structure and take advantage of it during learning and performance.

Figure 2 shows a MAXQ graph for this problem. This graph contains two kinds of nodes: Max nodes (indicated by triangles) and Q nodes (indicated by ovals). Max nodes with no children denote primitive actions in the domain; Max nodes with children represent subtasks. In this simple problem, there are five such subtasks: (a) Navigate(t) (move the taxi to target location t), (b) Get (move to the passenger's location and pick up the passenger), (c) Put (move to the passenger's destination and put down the passenger), (d) Refuel (move to F and Fillup), and (e) Root (perform the overall task of picking up and delivering the passenger). Notice that the Navigate task is shared by the Get, Put, and Refuel tasks.

The immediate children of each Max node are Q nodes. Each Q node represents an action that can be performed to achieve its parent's subtask. For example, the MaxGet node has a child QNavigateForGet which represents the action of navigating from the current state to the passenger's location. The distinction between Max nodes and Q nodes is critical to ensuring that subtasks can be shared and reused. Each Max node will learn the *context independent* expected cumulative reward of performing its subtask. For example, MaxNavigate(t) will estimate the expected cumulative reward of navigating from any state to one of the five target locations t . Each Q node will learn the *context dependent* expected cumulative reward of performing its subtask. For example, QNavigateForGet(t) will learn the expected cumulative reward of navigating to location t and then completing the Get task. On the other hand, QNavigateForPut(t) will learn the expected cumulative reward of navigating to location t and then completing the Put task. Both of these Q nodes will "ask" MaxNavigate(t) how much it will cost to get to location t , and they will use

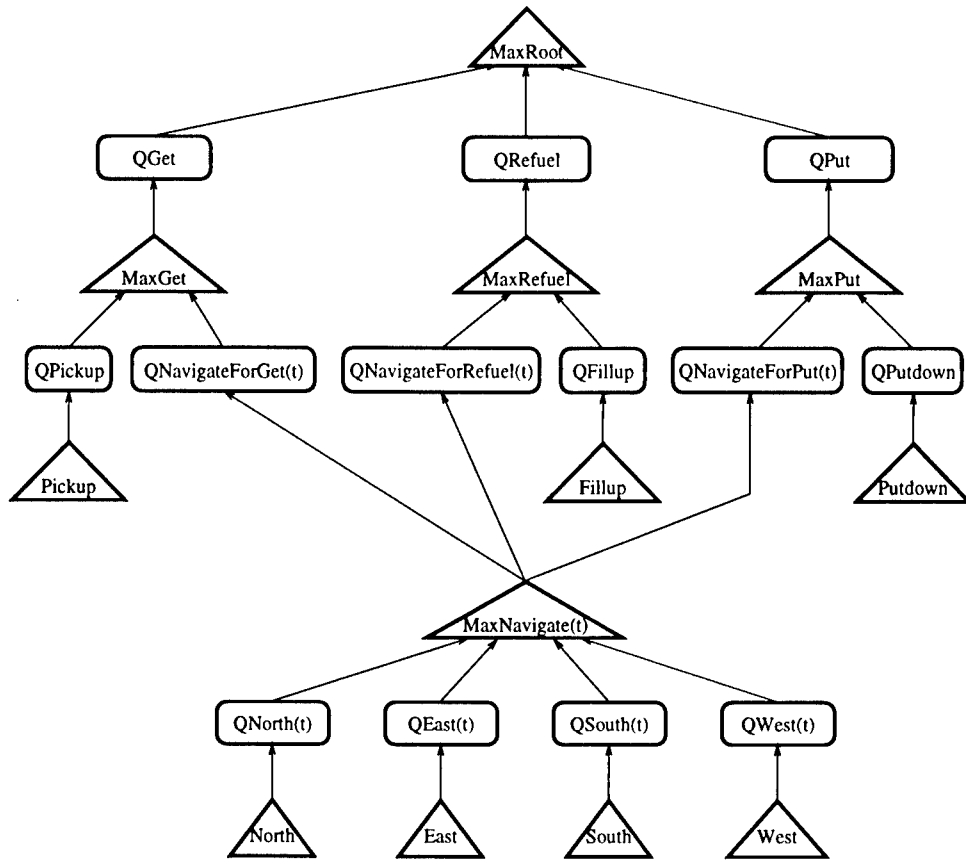


Figure 2: A MAXQ graph for the Taxi Domain

this to help them compute their Q values. The value function computed by MaxNavigate is context independent and can be shared by all three of its parent Q nodes.

In rest of the paper, we will say that Max node a is the child of Max node i if there is a Q node whose parent is i and whose child is a .

To define the semantics of the MAXQ graph more formally, let us suppose that the overall task is to solve a Markov Decision Problem (MDP) M defined over a set of states S and actions A with reward function $R(s'|s, a)$ (the reward received upon entering state s' after performing action a in state s) and transition probability function $P(s'|s, a)$ (the probability of entering state s' as a result of performing a in s). In this paper, we will assume that the MDP M defines an undiscounted stochastic shortest path problem. All of the results can be extended to the infinite-horizon discounted case.

Each Max node i corresponds to a separate subtask M_i . The children of Max node i are the actions of M_i . Each subtask M_i divides the set S of all states into two disjoint subsets: S_i and T_i . The set T_i is the set of *terminal states* for M_i .

Subtask M_i will terminate whenever the environment enters one of the states in T_i . A subset $G_i \subseteq T_i$ of the terminal states are the *goal states* of M_i . Below, we will discuss the details of defining a reward function that will encourage M_i to terminate in one of these goal states. Let us define π_i to be some (arbitrary) policy for subtask i . This policy “attempts” to get from any state in S_i to one of the goal states in G_i .

A *hierarchical policy* for a MAXQ graph is a set of policies $\pi = \{\pi_0, \dots, \pi_n\}$, one for each Max node, that indicate how each Max node should choose its actions. The hierarchical policy is executed the same way that subroutines are executed in ordinary programming languages. The Root policy chooses one of its child actions to perform, say, Get. The Get policy then chooses one of its child actions, say, Pickup. Then the Pickup action is executed, since it is a primitive. A Max node’s policy is executed until that Max node enters a terminating state, at which point, “control” returns to its parent Max node.

Therefore, we can view the MAXQ graph as a subroutine call graph. Like subroutines, Max nodes can be parameterized. In this graph, MaxNavigate takes one parameter,

t , which specifies which of the five locations (R, B, G, Y, F) is the target of the MaxNavigate. One way in which the graph is different from an ordinary program is that the children of each Max node are *unordered*. They can be called in any order, and a Max node can execute each of its children multiple times before it completes its subtask. The MAXQ graph is therefore a kind of incompletely-specified non-deterministic program. One result of learning will be to determine a policy for each Max node that tells how and when to invoke its children. This will make the MAXQ graph a completely-specified deterministic program (interacting with a non-deterministic environment).

Thus far, our formulation of the MAXQ method is essentially the same as the Feudal Q learning method of Dayan and Hinton (1993). However, an important improvement over Feudal Q learning is the ability to interpret the MAXQ graph as a representation of the value function for a hierarchical policy. Consider Max node i , and define $V_i^\pi(s)$ to be the expected cumulative reward for following the hierarchical policy π starting in state s until we enter some state in T_i . For a fixed hierarchical policy π , subtask M_i has a well-defined transition probability function $P_i^\pi(s'|s, a)$, which is the probability that the environment will move from state s to state s' when M_i executes action a . This probability is well defined, because the child M_a is executing a fixed policy π_a (as are all of its descendants). Hence, node i can treat action a as an atomic action. The immediate reward for node i of executing a will be the expected reward for node a of moving from the current state s to a terminal state in T_a according to policy π_a . This is denoted $V_a^\pi(s)$. Hence, we can write

$$V_i^\pi(s) = V_a^\pi(s) + \sum_{s'} P_i(s'|s, a) V_i^\pi(s'), \quad (1)$$

where $a = \pi_i(s)$. This gives us a recursive decomposition of the value function so that the value function of the root node is the value function of the entire MDP M and each subtask M_i is a separate MDP.

This recursive expression becomes more useful when we switch to the action-value (or "Q") representation of the value function. Define $Q_i^\pi(s, a)$ to be the expected cumulative reward for MDP M_i of performing action a in state s and then following the hierarchical policy π thereafter. Define the second term on the right-hand side of Eq. (1) to be $C_i^\pi(s, a)$, which we will call the *completion function*. This is the expected cumulative reward of *completing* MDP M_i following policy π after executing action a in state s . With these definitions, we can rewrite Eq. (1) as

$$Q_i^\pi(s, a) = V_a^\pi(s) + C_i^\pi(s, a) \quad (2)$$

where

$$V_i^\pi(s) = \begin{cases} Q_i^\pi(s, \pi_i(s)) & i \text{ composite} \\ \sum_{s'} P(s'|s, i) R(s'|s, i) & i \text{ primitive} \end{cases} \quad (3)$$

$$C_i^\pi(s, a) = \sum_{s'} P(s'|s, a) V_i^\pi(s') \quad (4)$$

These completely define the value-function semantics of the MAXQ hierarchy. Each Q node with parent i and child a stores the information $C_i^\pi(s, a)$ for each state s in S_i . Each Max node i returns the Q value of the child chosen by π_i .

To compute the value of a hierarchical policy π in state s , we begin at MaxRoot (node 0) and compute $Q_0^\pi(s, \pi_0(s))$. This requires that we ask our child node $a_1 = \pi_0(s)$ for its value $V_{a_1}^\pi(s)$. Our child recursively asks its child $a_2 = \pi_{a_1}(s)$ for its value, and so on until a leaf node a_n is reached. Let (a_1, a_2, \dots, a_n) be the path that was traversed through the MAXQ graph. Now leaf node a_n returns $V_{a_n}^\pi(s)$, to which its parent adds $C_{a_{n-1}}^\pi(s, a_n)$ and so on recursively. The value returned by MaxRoot is

$$V_0^\pi(s) = V_{a_n}^\pi(s) + C_{a_{n-1}}^\pi(s, a_n) + \dots + C_{a_1}^\pi(s, a_2) + C_0^\pi(s, a_1) \quad (5)$$

Figure 3 shows how the sequence of rewards r_1, r_2, \dots received from the primitive actions is decomposed hierarchically into the sum of the C terms.

3 Representation Theorems

Under what conditions can this hierarchy represent the value function of a fixed, hierarchical policy? We will say that a MAXQ graph is a *full-state* graph if separate $C_i^\pi(s, a)$ values are stored for each state $s \in S_i$. In most applications, including Figure 1, it will be desirable to introduce an abstraction function $X_i(s)$ that will provide a set of features that abstract essential information from the state. Each Q node will then store the function $C_i^\pi(X_i(s), a)$, with one value for each distinct abstract state $X_i(s)$.

For full-state graphs, it is easy to prove the following theorem by expanding Equations (2–4):

Theorem 1 *Let $\pi = \{\pi_i; i = 0, \dots, n\}$ be a hierarchical policy defined over a full-state MAXQ graph and let $i = 0$ be the root node of the graph. Then there exist values for C_i (for internal Max nodes) and V_i (for primitive, leaf Max nodes) such that $V_0(s)$ is the expected cumulative reward of following policy π in state s .*

A more important and difficult question is to understand the conditions under which an abstract-state MAXQ graph can exactly represent the value function of a hierarchical policy. The following theorem establishes one condition:

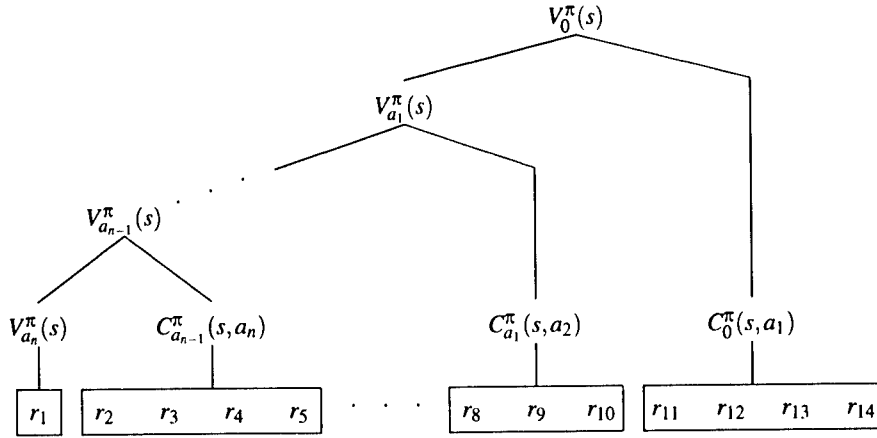


Figure 3: The MAXQ decomposition; r_1, \dots, r_{14} denote the sequence of rewards received from primitive actions at times $1, \dots, 14$.

Theorem 2 For all Max nodes i and actions a , let $\text{Result}_i^\pi(s, a) = \{s' | P_i^\pi(s' | s, a) > 0\}$ be the set of states that can result from applying abstract action a in state s at node i while following hierarchical policy π . If the following condition holds, then the MAXQ graph with abstraction functions $X_i(s, a)$ can represent the value function of any policy π whose value function can be represented by the MAXQ graph with no abstraction functions:

For all Max nodes i , actions a , states $s \in S_i$, and distinct states $s_1, s_2 \in \text{Result}_i^\pi(s, a)$ whenever $C_i^\pi(s_1, a) \neq C_i^\pi(s_2, a)$ it is the case that $X_i(s_1, a) \neq X_i(s_2, a)$

In other words, if an abstraction function X_i treats a pair of result states s_1 and s_2 as identical, then their un-abstrated values must be equal. Otherwise, the value function cannot be properly represented. The four children of MaxNavigate all satisfy this condition. The expected reward of completing the MaxNavigate action depends only on the current location of the taxi, the target location, and the amount of fuel remaining. If we are navigating to F (for refueling), for example, the expected reward does not depend on the source or destination locations.

The introduction of abstractions can create a *hierarchical credit assignment problem*. For example, in our implementation, we used only the taxi location and the target location to represent the C functions for QNorth, QSouth, QEast, and QWest. We wanted these nodes to learn a navigation policy that was independent of how much fuel remained. But this means that when the fuel is exhausted and a -20 penalty is received, these Q nodes cannot represent the reason for this penalty! This is the hierarchical credit assignment problem: to determine which node is responsible for a reward that is received. Our solution is for the designer of the MAXQ hierarchy to also *decompose the re-*

ward function. When each reward is generated, a marker is attached that indicates which Q nodes are potentially responsible for this reward. For the -20 empty fuel penalty, the QGet, QPut, and QRefuel nodes are held responsible, because their parent, MaxRoot, must compare their Q values to decide when to refuel to avoid the penalty. Their C functions must therefore be able to represent the rewards.

This requires a change to the decomposition equations. Let $R_i(s' | s, a)$ be the portion of the reward that is assigned to node i . Then we write the following:

$$C_i^\pi(s, a) = \sum_{s'} P(s' | s, a) [R_i(s' | s, a) + V_i^\pi(s')] \quad (6)$$

$$V_i^\pi(s) = \begin{cases} Q_i^\pi(s, \pi_i(s)) & i \text{ composite} \\ \sum_{s'} P(s' | s, i) R_i(s' | s, i) & i \text{ primitive} \end{cases} \quad (7)$$

In many domains, we believe it will be easy for the designer of the hierarchy to also decompose the reward function. However, an interesting problem for future research is to develop algorithms for autonomously solving the hierarchical credit assignment problem.

4 A Learning Algorithm

The preceding section has shown that the hierarchy can correctly represent the value function of any hierarchical policy if the full state is employed to represent the C_i function in each node i . Hence, we could apply Parr and Russell's HAM-Q algorithm to learn the best hierarchical policy. However, because we are committed to employing state abstractions, we have chosen instead to develop a reinforcement learning algorithm for finding a recursively optimal policy.

It turns out that in general there can be many different recursively optimal policies, and that some of them achieve

better expected rewards than others. The problem is that a subtask may have many policies that are locally optimal, but some of them are more useful than others for the overall task. For example, suppose we changed the taxi domain so that if the taxi hits a wall, the trial is terminated with a reward of -5 . Then for $\text{MaxNavigate}(t)$, if the target location t is more than 5 steps away, the locally optimal policy would be to hit a wall. This would not be part of any hierarchically optimal policy, however! Dayan and Hinton faced this same problem, and they solved it by providing a penalty of 10 points to subtask i for entering an undesired terminal state (i.e., a state in T_i but not in G_i). This has the proper effect, but in the MAXQ hierarchy, it causes the value function computed by the entire hierarchy to be incorrect, because it incorporates the (often non-zero) probability of receiving these terminal state penalties.

A better method is to define, for each Max node MDP M_i , a parallel Markov decision problem \tilde{M}_i with the same states, actions, and transition probabilities as M_i but with a second reward function \tilde{R}_i that is zero except for undesired terminal states, where it provides a large penalty. (We used a penalty of -100 points). Our learning algorithm will seek a locally optimal policy $\tilde{\pi}_i^*$ for \tilde{M}_i . However, it will also compute the value function for executing $\tilde{\pi}_i^*$ in the original MDP M_i , and this is the value that will be passed “up” the MAXQ hierarchy.

Specifically, our learning algorithm MAXQ-Q is a variant of Q learning that performs the following. At each composite Max node, we maintain two tables $C_i(s, a)$ and $\tilde{C}_i(s, a)$. The algorithm chooses an action a to perform according to its current exploration policy. It executes a , observes the resulting state s' and reward $R_i(s'|s, a)$, and computes the following:

$$a^* := \operatorname{argmax}_{a'} [\tilde{C}_i(s', a') + V_{a'}(s')] \quad (8)$$

$$\begin{aligned} \tilde{C}_i(s, a) &:= (1 - \alpha_t(i))\tilde{C}_i(s, a) + \alpha_t(i) \cdot \\ &\quad [\tilde{R}_i(s') + R_i(s'|s, a) + \tilde{C}_i(s', a^*) + V_{a^*}(s')] \end{aligned} \quad (9)$$

$$\begin{aligned} C_i(s, a) &:= (1 - \alpha_t(i))C_i(s, a) + \alpha_t(i) \cdot \\ &\quad [R_i(s'|s, a) + C_i(s', a^*) + V_{a^*}(s')] \end{aligned} \quad (10)$$

Here a^* is the best action in s' according to the current \tilde{C} and V values. Both \tilde{C} and C are updated using a^* . At each leaf node i , the update is slightly different:

$$V_i(s) := (1 - \alpha_t(i))V_i(s) + \alpha_t(i)R_i(s'|s, i). \quad (11)$$

The quantity $\alpha_t(i)$ is the learning rate for node i at time step t .

In order to prove convergence of this algorithm, we must make several assumptions. First, we must assume that all deterministic policies in MDP M are proper (i.e., they all terminate with probability 1). Second, we must assume that all locally optimal policies, $\tilde{\pi}_a^*$, give the same transition probability distribution $P_i^{\tilde{\pi}_a^*}(s'|s, a)$. This ensures that all locally optimal policies at node a give rise to the same MDP at any node i that is a parent of a . (A consequence of this assumption is that all recursively optimal policies will have the same value function.) Third, we must assume that $|V_i|$, $|C_i|$, and $|\tilde{C}_i|$ are bounded at all times (this is easy to enforce). Fourth, the exploration policy executed at each node i during learning must be a GLIE (greedy in the limit with infinite exploration) policy—that is, a policy that executes each action infinitely often in every state that is visited infinitely often, and that is greedy with respect to \tilde{Q}_i with probability 1. Finally, the learning rates $\alpha_t(i)$ must satisfy the usual conditions:

$$\lim_{T \rightarrow \infty} \sum_{t=1}^T \alpha_t(i) = \infty \quad \text{and} \quad \lim_{T \rightarrow \infty} \sum_{t=1}^T \alpha_t^2(i) < \infty \quad (12)$$

Theorem 3 *Under the assumptions listed above, with probability 1, MAXQ-Q will converge to a recursively optimal policy for MDP M consistent with MAXQ hierarchy H .*

Proof Sketch: The proof employs a stochastic approximation argument similar to those introduced to prove the convergence of Q learning and SARSA(0) (Jaakkola, Jordan, & Singh, 1994; Bertsekas & Tsitsiklis, 1996; Singh, Jaakkola, Littman, & Szepesvari, 1998). The proof is by induction on the levels of the tree, starting at the Max nodes all of whose children are primitive leaf nodes. At these “first-level” Max nodes, the standard results for Q learning can be applied to prove that the C_i values will converge with probability 1 to the optimal value function. Furthermore, because each node i is executing a GLIE exploration policy, the policy at these nodes will also converge with probability 1 to a locally optimal policy.

Now consider a Max node j all of whose children are either primitive nodes or “first-level” Max nodes. Define $P_j^t(s'|s, i)$ to be the transition probabilities observed by parent node j when it invokes child node i in state s at time t in the learning process. Because the first-level Max nodes are executing GLIE policies, $P_j^t(s'|s, i)$ will converge (with probability 1) to the state transitions $P_j^*(s'|s, i)$ that will be produced by any of the locally optimal policies for node i (by assumption, all of these locally optimally policies give the same state transition probabilities). This enables us to prove that node j also converges with probability 1 to the optimal C_j values and a locally-optimal policy. The

key is to decompose the error in any particular C_j backup into two terms. One term—corresponding to the difference between a sample backup (using the observed state transition) and a full Bellman backup (using $P_j^*(s'|s, i)$)—has expected value of zero. The other term—corresponding to the difference between doing a full Bellman backup using the current transition probabilities, $P_j^*(s'|s, i)$ and doing a full Bellman backup using the final transition probabilities $P_j^*(s'|s, i)$ —converges to zero with probability 1. By applying a stochastic approximation result (Proposition 4.5 from Bertsekas and Tsitsiklis, 1996), we can prove that node j will converge to a locally optimal policy. Hence, by induction, we can prove that the entire hierarchy converges to a recursively optimal policy. **End of Proof Sketch.**

There is one interesting method that can be employed to accelerate learning in the higher nodes of the graph. When an action a is chosen for Max node i in state s , the execution of a will move the environment through a series of states $s_1, \dots, s_k, s_{k+1} = s'$. If a was indeed the best action to choose in s_1 , then it should also be the best action to choose (at node i) in states s_2 through s_k . Hence, equations (9) and (10) can be applied in all of these states. This reflects an important difference between standard subroutine calls and the MAXQ hierarchy. In standard subroutines, there is a set of preconditions that must be true at the start of the subroutine. A partially-executed subroutine can often make these preconditions false, so that it is not possible to interrupt a subroutine and then call it again without first re-establishing the preconditions. In the MAXQ hierarchy, however, a Max node i can be invoked in any state $s \in S_i$, and it must “complete” execution of the task from that state onward. This means that the execution of the Max node can be interrupted and restarted with no change to the hierarchy.

We applied algorithm MAXQ-Q to the Taxi task using a tabular representation of the C functions. We employed state abstraction as follows. For the QNorth, QSouth, QEast, and QWest nodes, the C function ignores the passenger source and destination locations and the amount of fuel. The C function of QPickup ignores the passenger destination and fuel, but it must know the source location and taxi location in order to predict the effects of illegal Pickup actions. Similarly, QPutdown ignores the passenger source location and the fuel, and QFillup ignores the source and destination locations and the fuel. QNavigateForGet can represent its C function by a single value, because after a successful Navigate, only a Pickup remains to complete the Get action. The same is true for QNavigateForPut and QNavigateForRefuel. Because of the hierarchical credit assignment, QGet and QRefuel need to see the entire state, but QPut can ignore all of the state information, because

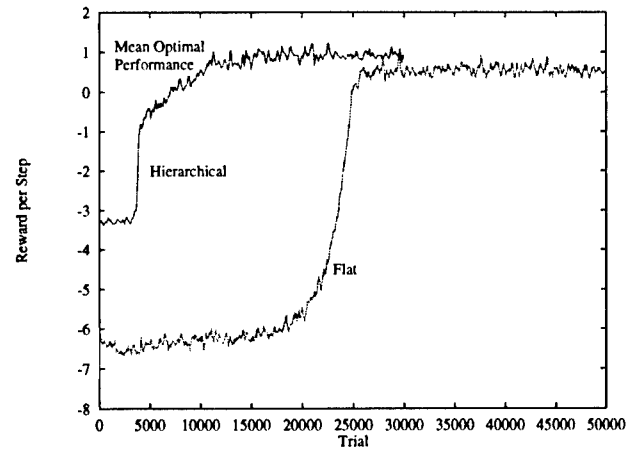


Figure 4: Online performance of flat and hierarchical Q learning on the Taxi task. Each curve is smoothed using a 200-trial moving average. The horizontal line shows the average performance of the optimal policy.

once it succeeds, the task is completed. All of these abstractions mean that instead of a set of seven 8,750-element Q functions (61,250 values) for flat Q learning, the MAXQ hierarchy requires only 18,253 values to represent the C functions.

Figure 4 compares the online performance of flat and hierarchical Q learning. For flat Q learning, we employed Boltzmann exploration with an initial temperature of 50. This was decreased by a factor of 0.997 after each successful trial. We experimented with many different cooling schedules, but we were unable to get flat Q learning to converge to the optimal policy within 50,000 trials. This was the fastest cooling schedule that was able to attain (at least briefly) the optimal expected reward. For hierarchical Q learning, we employed a separate temperature for each Max node. The starting temperature for all nodes was 50 except MaxRoot, which used 100. Each node decreased its temperature when it successfully reached a goal terminal state. MaxRoot was cooled by a factor of 0.9986, the second level Max nodes at 0.997, and MaxNavigate at 0.995. In all cases, a learning rate of $\alpha = 1$ was employed, since all actions and rewards are deterministic.

These cooling rates were chosen so that the lower Max nodes in the graph can become reasonably competent at their subtasks before the nodes higher in the graph try to learn. If care is not taken, a Max node i may conclude that a subtask a is very expensive (because the subtask has not yet learned a good policy), and therefore, it sets the C value for a very low. When this is combined with Boltzmann exploration, the result is that the subtask may never be tried again. Hence, we only performed an update for a Q node if that node completed its subtask with an average absolute

Bellman error per step of less than 0.2. (This parameter was not tuned at all.)

Figure 4 shows that the hierarchical method is able to learn the task much faster and achieve a higher level of performance than flat Q learning. Of course, both methods could be improved by employing techniques for accelerating Q learning, such as eligibility traces (e.g., Peng & Williams, 1996).

5 Non-Hierarchical Execution

We have shown that the MAXQ hierarchy can learn an optimal policy for an MDP if that policy is a recursively optimal hierarchical. However, there are situations in which the optimal policy is almost—but not quite—hierarchical. For example, consider a modified Taxi task (the “fickle Taxi problem”) in which as soon as the taxi picks up the passenger and moves one square, the passenger can randomly change the destination with probability 0.3. This change comes after the hierarchical policy has committed to executing `QNavigateForPut(t)` for the original destination. As a result, the `MaxNavigate` subtask will take the taxi to the old destination. Then control will return to `MaxPut`, which will invoke `QNavigateForPut` to move the taxi to the new destination.

Such “almost hierarchical” MDP’s raise the question of whether there is a way to convert a recursively-optimal hierarchical policy into an optimal non-hierarchical policy.

To answer this question, we implemented the Fickle Taxi domain. We removed all aspects of fuel from the domain so that we could figure out the optimal policy and hand-code it. Figure 5 compares the performance of flat Q learning and hierarchical Q learning on this modified task. The optimal policy can achieve an average reward per step of 1.172; but the best hierarchical policy (compatible with the MAXQ graph of Figure 2) can only achieve 1.002. Hierarchical learning with MAXQ-Q is able to attain this level rapidly. Flat Q learning approaches the optimum, but does not reach it within 10,000 trials. We tuned each algorithm to optimize its performance. We employed a learning rate of 0.35 and decayed the initial temperature of 50.0 by a factor of .460 (for flat Q) and .211 (for hierarchical Q) whenever a goal terminal state was reached.

An alternative to hierarchical execution of the MAXQ graph is *polling execution*, as first suggested by Kaelbling in her (1993) Hierarchical Distance to Goal method. In the polling approach to MAXQ, each action is chosen by starting at `MaxRoot` and computing the path (from root to leaf) with the highest Q value. The primitive action at the end of this path is then executed, and the process is repeated. This

is equivalent to computing the one-step greedy lookahead policy given the current value function. If the hierarchical policy is not optimal, then this one-step greedy policy will be closer to an optimal policy, because it corresponds to one step of policy improvement in the policy iteration algorithm (Bertsekas, 1995). This informally proves the following:

Theorem 4 *For all states s , the value of the policy computed by polling execution of the MAXQ hierarchy is \geq the value of the policy computed by hierarchical execution.*

Hence, polling execution of a MAXQ graph can produce a non-hierarchical policy that is better than the hierarchical policy represented by the graph.

We tested this on the Fickle Taxi task by first training the MAXQ hierarchy by MAXQ-Q for 1000 trials and then continuing the training with polling execution. Figure 6 shows that there is an initial loss of performance when we switch to polling execution. This is because during hierarchical training, the more abstract Q nodes in the graph have only learned their C values well in states where they were frequently executed. Under polling, they are now executed in other states as well, and they rapidly learn the correct values so that performance is able to reach the level of the optimal non-hierarchical policy. In this domain, polling execution of the best hierarchical policy can produce the optimal policy.

6 Concluding Remarks

This paper has defined the MAXQ value function decomposition for hierarchical reinforcement learning. The paper has shown that the MAXQ graph can represent the value function of any hierarchical policy implemented by the graph. A learning algorithm based on Q learning was introduced, proved to converge, and shown experimentally to perform much better than ordinary, non-hierarchical Q learning.

The most important aspect of the MAXQ method is the separation between the context-independent policy and value function (represented by the `Max` nodes) and the context-dependent value function (represented by the `Q` nodes). This permits the value functions of subtasks to be learned independent of their context, and this enhances the reusability of the subtasks and makes it easier to employ state abstraction within the subtasks. However, optimality of the learned policy is lost in general, and hierarchical credit-assignment problems may be introduced. Fortunately, the ability of the MAXQ hierarchy to represent the value function of the hierarchical policy permits the non-hierarchical execution of a one-step greedy policy that is better than the

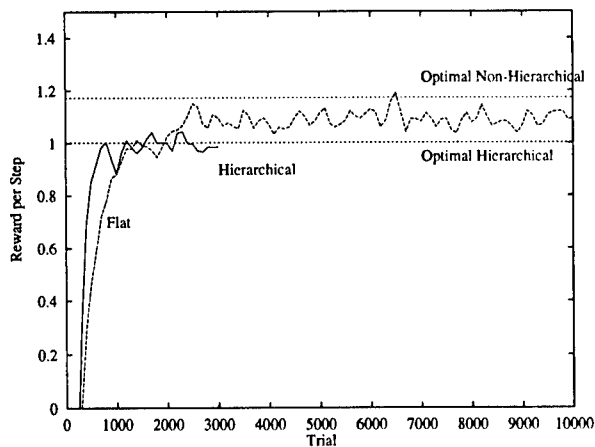


Figure 5: Online performance of flat and hierarchical Q learning on the Fickle Taxi task. Each curve is the average of 10 runs; the returns from each run were smoothed by a 200-trial moving average.

hierarchical policy.

Acknowledgements. The author thanks Eric Chown for many helpful discussions of this work and Valentina Bayer, William Langford, and Wesley Pinchot for helpful comments on an earlier draft. The support of ONR grant N00014-95-1-0557 and of NSF grant IRI-9626584 is gratefully acknowledged.

References

- Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Athena Scientific, Belmont, MA.
- Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.
- Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. *NIPS*, 5, pp. 271–278. Morgan Kaufmann, San Francisco, CA.
- Dean, T., & Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. Tech. rep. CS-95-10, Dept. of Computer Science, Brown University, Providence, Rhode Island.
- Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neur. Comp.*, 6(6), 1185–1201.
- Kaelbling, L. P. (1993). Hierarchical reinforcement learning: Preliminary results. *ICML-93*, pp. 167–173. San Francisco, CA. Morgan Kaufmann.
- Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *NIPS*, Vol. 10. Cambridge, MA. MIT Press.

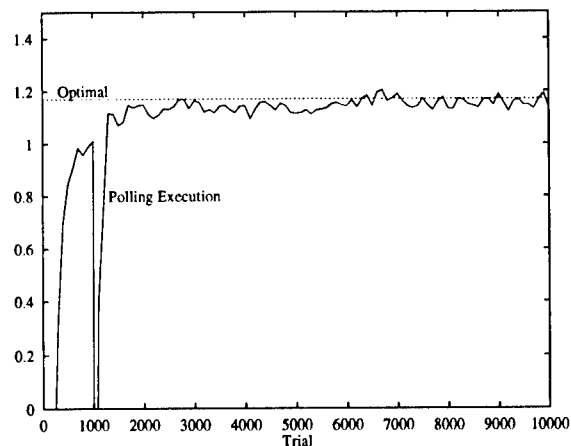


Figure 6: Online performance on the Fickle Taxi task. The first 1000 trials are trained hierarchically. The remaining trials are trained while polling.

Peng, J., & Williams, R. J. (1996). Incremental multi-step Q-learning. *Mach. Learn.*, 22, 283–290.

Singh, S., Jaakkola, T., Littman, M. L., & Szepesvari, C. (1998). Convergence results for single-step on-policy reinforcement-learning algorithms. Tech. rep., University of Colorado, Dept. Comp. Sci.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Mach. Learn.*, 8, 323–339.

Sutton, R. S., Precup, D., & Singh, S. (1998). Between MDPs and Semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. Tech. rep., University of Mass., Dept. Comp. Inf. Sci., Amherst, MA.

A Process-Oriented Heuristic for Model Selection

Pedro Domingos

Artificial Intelligence Group
 Instituto Superior Técnico
 Lisbon 1096, Portugal
 pedrod@gia.ist.utl.pt

Abstract

Current methods to avoid overfitting are either data-oriented (using separate data for validation) or representation-oriented (penalizing complexity in the model). This paper proposes process-oriented evaluation, where a model's expected generalization error is computed as a function of the search process that led to it. The paper develops the necessary theoretical framework, and applies it to one type of learning: rule induction. A process-oriented version of the CN2 rule learner is empirically compared with the default CN2. The process-oriented version is more accurate in a large majority of the datasets, with high significance, and also produces simpler models. Experiments in artificial domains suggest that process-oriented evaluation is particularly useful in high-dimensional domains.

1 INTRODUCTION

Overfitting avoidance is often considered the central problem of machine learning (e.g., (Cheeseman & Oldford, 1994)). If a learner is sufficiently powerful, it must guard against selecting a model that fits the training data well but captures the underlying phenomenon poorly. Current methods to address this problem fall into two broad categories. *Data-oriented evaluation* uses separate data to learn and validate models, and includes methods like cross-validation (Breiman, Friedman, Olshen & Stone, 1984; Stone, 1974), the bootstrap (Efron & Tibshirani, 1993), and reduced-error pruning (Brunk & Pazzani, 1991). It has several disadvantages: it is often computationally

intensive, reduces the data available for learning, can be unreliable if the validation set is small, and is itself prone to overfitting if a large number of models is compared (Ng, 1997). *Representation-oriented evaluation* seeks to avoid these problems by using the same data for training and validation, but *a priori* penalizing some models as more likely to overfit. Bayesian approaches in general fall into this category (Cheeseman, 1990; MacKay, 1992). Representation-oriented measures typically contain two terms, one reflecting fit to the data, and one penalizing model complexity (Akaike, 1978; Schwarz, 1978; Wallace & Boulton, 1968; Rissanen, 1978; Moody, 1992). This approach is only appropriate when the simpler models are truly the more accurate ones, and there is mounting evidence that this is typically not the case ((Domingos, 1998; Domingos, 1997; Schuurmans, Ungar & Foster, 1997; Lawrence, Giles & Tsoi, 1997; Webb, 1996; Schaffer, 1993; Murphy & Pazzani, 1994), etc.). Structural risk minimization (Vapnik, 1995) and PAC learning (Kearns & Vazirani, 1994) are representation-oriented methods that seek to bound the difference between training and generalization error using a function of the model space's (effective) dimension. This typically produces bounds that are overly broad, and requires severely restricting the model space.

In this paper we argue that representation-oriented evaluation has these limitations because it only considers the learner's model space, and not its search process. A learner with an unlimited model space can avoid overfitting as long as it attempts only a limited number of hypotheses (even if it is not possible *a priori* to predict which). If these hypotheses are correlated, the chance of overfitting is further reduced. Given the sequence of hypotheses that a learner attempts, it is possible to estimate the generalization error of the "current best" hypothesis taking into account the process that led to it. Intuitively, the more hypotheses

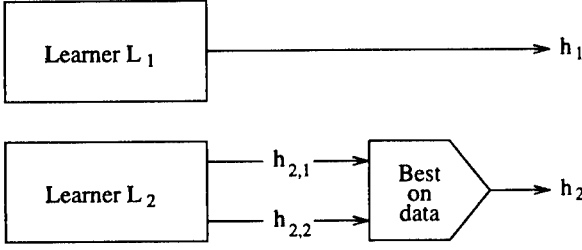


Figure 1: A simple example of an overfitting avoidance problem.

that have been attempted and the less correlated they are, the higher the generalization error we expect for a given training-set error. This paper begins to develop this approach, which we will call *process-oriented evaluation* (POE for short). The basic theoretical framework is presented, and then applied to the standard “separate and conquer” rule induction process (Clark & Niblett, 1989). An empirical study demonstrates the effectiveness of POE. The paper concludes with sections on related and future work.

2 PROCESS-ORIENTED EVALUATION

Consider the simplest example of an overfitting avoidance problem, in a classification context. Suppose learner L_1 consists of drawing one hypothesis at random from some model space and returning it, and learner L_2 consists of drawing two hypotheses at random (independently) from the same model space as L_1 , and returning the one with lowest error on a training sample S . This situation is shown schematically in Figure 1. Let h_1 be the hypothesis returned by L_1 , h_2 the hypothesis returned by L_2 , n the number of examples in S , and e_i the number of examples h_i misclassifies. The goal is to choose the hypothesis with lowest true error ϵ_i (i.e., ϵ_i is the probability of h_i misclassifying an example, given the true example distribution). Suppose $n = 100$, $e_1 = 12$, and $e_2 = 11$. Should we prefer h_1 or h_2 ? According to the maximum likelihood principle (DeGroot, 1986), $\hat{\epsilon}_1 = 0.12$ and $\hat{\epsilon}_2 = 0.11$, so h_2 should be chosen. Assuming the two hypotheses have the same complexity or prior probability, representation-oriented evaluation would give the same answer. However, L_2 had two opportunities to draw a hypothesis with low training error, and so the probability of e_2 being low merely by chance is higher than for e_1 . Thus h_2 may in fact have a higher true error rate than h_1 .

This notion can be quantified. If a hypothesis h ’s true error rate is ϵ and S consists of n independently drawn examples, the number of errors e committed by h on S is a binomially distributed variable with parameters n and ϵ :

$$p(e|n, \epsilon) = b(e|n, \epsilon) = \binom{n}{e} \epsilon^e (1 - \epsilon)^{n-e} \quad (1)$$

Let $B(e|n, \epsilon)$ be the probability that the number of errors is greater than e :

$$B(e|n, \epsilon) = \sum_{i=e+1}^n b(i|n, \epsilon) \quad (2)$$

Notice that this notation is the opposite of the usual notation for a cumulative distribution function (i.e., $B(e|n, \epsilon) = 1 - \text{Binomial.cdf}(e|n, \epsilon)$). It will be more convenient for what follows.

The probability of h_1 misclassifying e_1 examples is $p(e_1|n, \epsilon_1) = b(e_1|n, \epsilon_1)$. This can be used with Bayes’s theorem to compute the expected value of ϵ_1 given n and e_1 , $E[\epsilon_1|n, e_1]$. By finding a similar expression for $p(e_2|n, \epsilon_2)$, we can compute $E[\epsilon_2|n, e_2]$ and choose the hypothesis with lowest expected error. Let the two hypotheses drawn by L_2 be $h_{2,1}$ and $h_{2,2}$ (with true errors $\epsilon_{2,1}$ and $\epsilon_{2,2}$ respectively, and numbers of training errors $e_{2,1}$ and $e_{2,2}$). From these, L_2 chooses the one with lowest training error (i.e., $h_2 = h_{2,j}$, where $j = \text{argmin}_{i \in \{1,2\}} e_{2,i}$). Then the probability of L_2 returning a hypothesis h_2 that misclassifies e_2 training examples is the probability that $h_{2,1}$ misclassifies e_2 training examples and $h_{2,2}$ misclassifies more, or vice-versa, or both $h_{2,1}$ and $h_{2,2}$ misclassify e_2 examples:

$$\begin{aligned} p(e_2|n, \epsilon_2) &= b(e_2|n, \epsilon_{2,1})B(e_2|n, \epsilon_{2,2}) \\ &\quad + B(e_2|n, \epsilon_{2,1})b(e_2|n, \epsilon_{2,2}) \\ &\quad + b(e_2|n, \epsilon_{2,1})b(e_2|n, \epsilon_{2,2}) \end{aligned} \quad (3)$$

Our goal is to use this equation to compute the expected value of ϵ_2 . We are hindered by the fact that in addition to e_2 (whether it is $e_{2,1}$ or $e_{2,2}$) the equation contains another unknown parameter (whichever $\epsilon_{2,i}$ is not ϵ_2). Since we are not interested in $\epsilon_{2,1}$ or $\epsilon_{2,2}$ *per se*, but only in the effect on ϵ_2 of trying two hypotheses instead of one, we propose the following heuristic: assume that $\epsilon_{2,1} = \epsilon_{2,2} = \epsilon_2$. This approximation will be good if $\epsilon_{2,1}$ and $\epsilon_{2,2}$ are similar, and poor if they are very different. However, this heuristic

may yield good results even in the latter case, because a close approximation of $E[\epsilon_2|n, e_2]$ is not required; all that is required is that $E[\epsilon_2|n, e_2] > E[\epsilon_1|n, e_1]$ iff $\epsilon_2 > \epsilon_1$, which is a much weaker condition (Domingos & Pazzani, 1997). If $\epsilon_{2,1} = \epsilon_{2,2} = \epsilon_2$ Equation 3 becomes:

$$\begin{aligned} p(e_2|n, \epsilon_2) &= b(e_2|n, \epsilon_2)B(e_2|n, \epsilon_2) \\ &\quad + B(e_2|n, \epsilon_2)b(e_2|n, \epsilon_2) \\ &\quad + b(e_2|n, \epsilon_2)b(e_2|n, \epsilon_2) \\ &= [B(e_2|n, \epsilon_2) + b(e_2|n, \epsilon_2)]^2 \\ &\quad - B^2(e_2|n, \epsilon_2) \\ &= B^2(e_2 - 1|n, \epsilon_2) - B^2(e_2|n, \epsilon_2) \quad (4) \end{aligned}$$

Applying Bayes's theorem:

$$p(\epsilon_2|n, e_2) \propto p(\epsilon_2)p(e_2|n, \epsilon_2) \quad (5)$$

$p(\epsilon_2)$ can be used to incorporate prior beliefs about the error rate of the hypotheses considered by L_2 . Here it will simply be assumed uniform.¹

$$p(\epsilon_2|n, e_2) \propto p(e_2|n, \epsilon_2) \quad (6)$$

The expected value of ϵ_2 can now be computed by integration:

$$E[\epsilon_2|n, e_2] = \frac{\int_0^1 \epsilon_2 p(e_2|n, \epsilon_2) d\epsilon_2}{\int_0^1 p(e_2|n, \epsilon_2) d\epsilon_2} \quad (7)$$

Doing this for $e_2 = 11$, $n = 100$ results in $E[\epsilon_2|n, e_2] = 0.134$. A similar treatment for e_1 , using $e_1 = 12$, $n = 100$ and $p(e_1|n, \epsilon_1) = b(e_1|n, \epsilon_1)$, yields $E[\epsilon_1|n, e_1] = 0.127$. Thus the hypothesis output by L_1 would be preferred, even though L_2 's has a lower training error.

Equation 4 can be readily generalized to a learner L_m that draws m hypotheses at random and chooses the one with lowest training error:

¹This is an unrealistic assumption, and is made solely for the sake of simplicity. As the following sections show, the proposed method can be effective even when this assumption is used. This can be attributed to the fact that, except for very small sample sizes and/or very extreme priors, the effect of the likelihood term $p(e_2|n, \epsilon_2)$ will easily dominate the prior's. In any case, a version of POE using beta priors is currently being implemented.

$$\begin{aligned} p(\epsilon_m|n, e_m) &\propto p(e_m|n, \epsilon_m) \\ &= B^m(e_m - 1|n, \epsilon_m) - B^m(e_m|n, \epsilon_m) \end{aligned} \quad (8)$$

Notice that this formula makes intuitive sense: as m increases, the mass of probability is shifted to higher and higher ϵ_m 's; but as n increases, higher and higher m 's are needed to make this happen to the same degree. To see this, consider the binomial expansion

$$\begin{aligned} B^m(e_m - 1|n, \epsilon_m) &= [B(e_m|n, \epsilon_m) + b(e_m|n, \epsilon_m)]^m \\ &= B^m(e_m|n, \epsilon_m) + mB^{m-1}(e_m|n, \epsilon_m)b(e_m|n, \epsilon_m) \\ &\quad + \frac{m(m-1)}{2}B^{m-2}(e_m|n, \epsilon_m)b^2(e_m|n, \epsilon_m) + \dots \end{aligned} \quad (9)$$

and consider that, for all but the smallest sample sizes, $B(e_m|n, \epsilon_m) \gg b(e_m|n, \epsilon_m)$. Thus:

$$\begin{aligned} p(\epsilon_m|n, e_m) &\propto p(e_m|n, \epsilon_m) \\ &= B^m(e_m - 1|n, \epsilon_m) - B^m(e_m|n, \epsilon_m) \\ &\simeq mb(e_m|n, \epsilon_m)B^{m-1}(e_m|n, \epsilon_m) \end{aligned} \quad (10)$$

When $m = 1$, this reduces to $b(e_m|n, \epsilon_m)$, as expected. When $m = 2$, $b(e_m|n, \epsilon_m)$ is multiplied by a constant and by $B(e_m|n, \epsilon_m)$. Since the latter is a function that increases monotonically with ϵ_m for a given n and e_m , the effect of this is to decrease the probability of lower ϵ_m 's and increase the probability of higher ones, and thus to increase the expected ϵ_m . As m increases, $b(e_m|n, \epsilon_m)$ is multiplied by higher and higher powers of $B(e_m|n, \epsilon_m)$. This further decreases the probability of low ϵ_m 's and increases the probability of high ones, leading to an ever-increasing expected ϵ_m . As an example, Figure 2 shows $b(25|50, \epsilon_m)$ (magnified by a factor of five) and several powers of $B(25|50, \epsilon_m)$. The resulting $E[\epsilon_m|50, 25]$ (not shown) has a roughly similar shape to $b(25|50, \epsilon_m)$, but shifts rightward in step with $B(25|50, \epsilon_m)$. For larger n , the same process takes place, but $b(e_m|n, \epsilon_m)$ is more sharply peaked, $B(e_m|n, \epsilon_m)$ also transitions from values close to zero to values close to one more sharply, and the advance of $B^m(e_m|n, \epsilon_m)$ to the right becomes correspondingly slower (since, for any $0 < k < y < 1$, as $y \rightarrow 1$ with

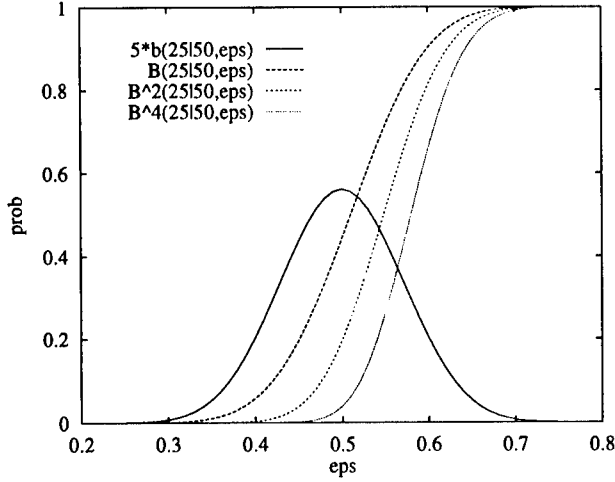


Figure 2: Variation of $b(e_m|n, \epsilon_m)$ and powers of $B(e_m|n, \epsilon_m)$ with ϵ_m for $n = 50, e_m = n/2$.

k held constant higher and higher m 's are needed to make $y^m \leq k$). This can be seen by comparing Figure 2 with Figure 3, which shows the corresponding plots for $n = 500$.

Equation 8 still assumes that all m hypotheses drawn are independent, but it can be further generalized to include the dependent case:

$$\begin{aligned} p(\epsilon_m|n, e_m) &\propto p(e_m|n, \epsilon_m) \\ &= p(\forall_{1 \leq i \leq m} e_{m,i} \geq e_m|n, \epsilon_m) \\ &\quad - p(\forall_{1 \leq i \leq m} e_{m,i} > e_m|n, \epsilon_m) \end{aligned} \quad (11)$$

Evaluating this expression when high-order dependencies are present will generally not be feasible, but the standard Bayesian network approach (Heckerman, 1996) is applicable here: the number of training errors $e_{m,i}$ of each hypothesis $h_{m,i}$ generated by L_m can be viewed as a node in a Bayesian network, whose parents are the training errors of the hypotheses $h_{m,j}$ it is primarily dependent on. For example, in many greedy search processes (e.g., standard decision tree induction), if $h_{m,3}$ was derived from $h_{m,2}$, which in turn was derived from $h_{m,1}$, $e_{m,3}$ will be approximately independent of $e_{m,1}$ given $e_{m,2}$. In general, the Bayesian network for a given learning process will have the DAG (directed acyclic graph) of the search process itself as a subgraph (e.g., in a greedy search each node $e_{m,i}$ will have arcs to the training errors of the hypotheses that were generated from $h_{m,i}$). If $\text{par}(e_{m,i})$ are the parents of $e_{m,i}$ in the Bayesian network, Equation 11 above reduces to:

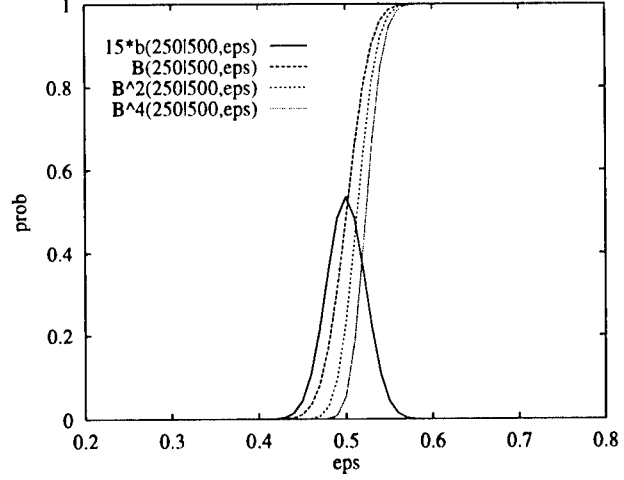


Figure 3: Variation of $b(e_m|n, \epsilon_m)$ and powers of $B(e_m|n, \epsilon_m)$ with ϵ_m for $n = 500, e_m = n/2$.

$$\begin{aligned} p(\epsilon_m|n, e_m) &\propto p(e_m|n, \epsilon_m) = \\ &\prod_{i=1}^m p(e_{m,i} \geq e_m|n, \epsilon_m, \forall_{e_{m,j} \in \text{par}(e_{m,i})} e_{m,j} \geq e_m) \\ &\quad - \prod_{i=1}^m p(e_{m,i} > e_m|n, \epsilon_m, \forall_{e_{m,j} \in \text{par}(e_{m,i})} e_{m,j} > e_m) \end{aligned} \quad (12)$$

L_1 and L_2 above were considered to be different learners, but they can equally well be considered different stages of the same learner. For example, L_2 can take the hypothesis output by L_1 as its own first hypothesis. More generally, L_m can be the result of continuing the search of learner L_k ($k < m$) with $m-k$ more hypotheses. Thus this framework can be applied to problems like decision tree and rule pruning, to which we now turn.

3 AN APPLICATION: RULE INDUCTION

Most rule induction systems employ a set covering or “separate and conquer” search strategy (Michalski, 1983; Clark & Niblett, 1989). Rules are induced one at a time, and each rule starts with a training set composed of the examples not covered by any previous rules. A rule is induced by adding conditions one at a time, starting with none (i.e., the rule initially covers the entire instance space). The next condition to add is chosen by attempting all possible conditions. Con-

ditions on symbolic attributes are typically of the form $a_i = v_{ij}$, where v_{ij} is a possible value of attribute a_i . Conditions on numeric attributes are typically of the form $a_i \leq v_{ij}$ or $a_i > v_{ij}$, where the thresholds v_{ij} are usually values of the attribute that appear in the training set. In the beam search process used by many rule learners, at each step the best b versions of the rule according to some evaluation function are selected for further specialization. AQ (Michalski, 1983) continues adding conditions until the rule is "pure" (i.e., until it covers examples of only one class). This can lead to severe overfitting. The latest version of the CN2 system (Clark & Boswell, 1991) uses a simple and effective Bayesian method to combat this: induction of a rule stops when no specialization improves its error rate, and the latter is computed using a *Laplace correction* or *m-estimate*. If n_r is the number of examples covered by a rule r , and e_r is the number of those examples it misclassifies, the conventional estimate of the rule's error rate is e_r/n_r , but its m-estimate is:

$$\hat{\epsilon}_r = \frac{e_r + m\epsilon_0}{n_r + m} \quad (13)$$

where ϵ_0 is the rule's *a priori* error, which CN2 takes to be the error obtained by random guessing if all classes are equally likely: $\epsilon_0 = (c - 1)/c$, where c is the number of classes. This prior value is given a weight of m examples (i.e., the behavior of Equation 13 is equivalent to having m additional examples covered by the rule, one of each class). CN2 uses $m=c$. As conditions are added, the rule covers fewer and fewer examples, and $\hat{\epsilon}_r$ tends to ϵ_0 . Thus a rule making more misclassifications may be preferred if it covers more examples, causing induction to stop earlier and reducing overfitting. Clark and Boswell (Clark & Boswell, 1991) found this version of CN2 to be more accurate than C4.5 (Quinlan, 1993) on 10 of the 12 benchmark datasets they used for testing. However, this scheme ignores that, as more and more conditions are attempted, the probability of finding one that appears to reduce the rule's error merely by chance increases. This will lead the m-estimate to underestimate the chosen condition's true error, and CN2 to overfit. The upward correction made to ϵ_r should increase with the number of conditions attempted. The process-oriented evaluation framework described in the previous section allows us to do this in a systematic way.

Let each hypothesis be one version of the rule attempted during the beam search. The main change to Equation 8 required is to take into account that different versions of a rule will cover different numbers

of training examples. In other words, n is now a function of the hypothesis, and the hypothesis with lowest e_i/n_i is chosen. Let $\vec{n}_m = (n_1, \dots, n_i, \dots, n_m)$, where n_i is the number of examples covered by rule version i , and let $\hat{\epsilon}_m = \min_{1 \leq i \leq m} \{e_i/n_i\}$ be the lowest training-set error rate found so far. Equation 8 becomes:

$$p(\epsilon_m | \vec{n}_m, \hat{\epsilon}_m) \propto p(\hat{\epsilon}_m | \vec{n}_m, \epsilon_m) = \prod_{i=1}^m B(n_i \hat{\epsilon}_m - 1 | n_i, \epsilon_m) - \prod_{i=1}^m B(n_i \hat{\epsilon}_m | n_i, \epsilon_m) \quad (14)$$

This equation does not need to be computed for every rule version generated during the beam search, but only once for each round. One round consists of generating every possible one-step specialization of each rule version in the beam, and selecting the b best. Thus, if there are a attributes and v is the maximum number of values of any attribute (in the worst case, $v = n$ for numeric attributes), one round corresponds to $O(bav)$ rule versions. Let m_k be the total number of rule versions generated up to, and including, round k . Round 1 consists of the initial rule with no conditions, and $m_1 = 1$. Induction stops when $E[\epsilon_{m_k} | \vec{n}_{m_k}, \hat{\epsilon}_{m_k}] \geq E[\epsilon_{m_{k-1}} | \vec{n}_{m_{k-1}}, \hat{\epsilon}_{m_{k-1}}]$, for $k > 1$.

Equation 14 is of course only a first approximation. Many other aspects of the rule induction process can be taken into account using Equation 12, and making approximations as needed for computational efficiency. A version of CN2 that takes into account the dependence between each rule version and its parent (i.e., the rule version it specializes by one condition) is currently being implemented.

4 EMPIRICAL STUDY

In order to test the effectiveness of process-oriented evaluation, default and process-oriented versions of CN2 were compared on the benchmark datasets previously used by Clark and Boswell (1991).² The process-oriented version was implemented by adding the necessary facilities to the CN2 source code. Numerical integration (Equation 7) was performed using Simpson's rule, and $B(e|n, \epsilon)$ (Equation 2) was computed using the incomplete beta function (Press, Teukolsky, Vetterling & Flannery, 1992). Integrating Equation 14 every time $E[\epsilon_{m_k} | \vec{n}_{m_k}, \hat{\epsilon}_{m_k}]$ needs to be computed (once

²With the exception of pole-and-cart, which is not available in the UCI repository (Merz, Murphy & Aha, 1997).

per round) would generally significantly slow down the rule induction process. Instead, it was approximated by:

$$p(\epsilon_m | \bar{n}, \hat{\epsilon}_m) \propto p(\hat{\epsilon}_m | \bar{n}, \epsilon_m) = B^m(\bar{n}\hat{\epsilon}_m - 1 | \bar{n}, \epsilon_m) - B^m(\bar{n}\hat{\epsilon}_m | \bar{n}, \epsilon_m) \quad (15)$$

where $\bar{n} = \frac{1}{m} \sum_{i=1}^m n_i$. This replaces each of the products with a single-step computation, speeding up evaluation by $O(m)$. CN2's Laplace estimates are still used to choose the best b specializations in each round. This is preferable to using uncorrected estimates, since as implemented POE has no preference between hypotheses within the same round, and this is also a factor in avoiding overfitting. However, the Laplace correction distorts the values used by Equation 15. This will be particularly pronounced when there are many classes, since CN2 uses $m = c$. In order to minimize this problem, $m = 2$ was used with POE.³

The experimental procedure of (Clark & Boswell, 1991) was followed. Each dataset was randomly divided into 67% for training and 33% for testing, and the error rate and theory size (total number of conditions) were measured for default CN2 and CN2-POE. This was repeated 20 times. The average results and their standard deviations are shown in Table 1.⁴

POE reduces CN2's error rate in 8 of the 11 datasets. Using a sign test, these results are significant at the 4% level. In other words, POE improves CN2 with high confidence. It also produces simpler rule sets in all but two of the datasets. With the approximation used, POE did not noticeably increase CN2's running time. This is also due to the fact that POE tends to make induction stop sooner than in default CN2, as evinced by the theory size results.

While these results are encouraging, they do not necessarily prove that CN2-POE reduces overfitting by taking into account the increasing number of rule versions generated as search progresses. If this is indeed what is taking place, the difference in error between default CN2 and CN2-POE ($error_{CN2} - error_{CN2-POE}$) should increase with the dataset's number of at-

³Simply changing $m = c$ to $m = 2$ in default CN2 does not change its performance on the datasets used.

⁴There are some differences between CN2's results and those reported in (Clark & Boswell, 1991). This may be due to the fact that the default version of CN2 uses a beam size of 5, whereas Clark and Boswell used $b = 20$. The distribution version of CN2 may also differ from the one used in (Clark & Boswell, 1991).

tributes, since this will increase the number of rule versions generated in each round. In order to test this hypothesis, experiments were carried out in artificial domains. Concepts defined as Boolean functions in disjunctive normal form were used as targets. The datasets were composed of 100 training examples and 1000 test examples described by a variable number of attributes a . The number of literals d in each disjunct was generated at random, with a mean of $d = 5$ and a variance of $5 \times (1 - \frac{5}{a})$. This is obtained by including each literal in the disjunct with probability $\frac{5}{a}$. Literals were negated or not with equal probability. The number of disjuncts was set to $2^{d-1} = 16$, which ensures the concept covers roughly half the instance space. Equal numbers of positive and negative examples were included in the dataset, and positive examples were divided evenly among disjuncts. In each run a different target concept was used. One hundred runs were conducted for each value of a between 10 and 100 (at intervals of 5), and the correlation between ($error_{CN2} - error_{CN2-POE}$) and a was measured. This was found to be highly positive ($\rho = 0.66$), confirming our hypothesis.

5 RELATED WORK

The literature on model selection and error estimation is very large, and we will not attempt to review it here. The incompleteness of representation-oriented evaluation was noted 20 years ago by Pearl (1978):

It would, therefore, be more appropriate to connect credibility with the nature of the selection procedure rather than with properties of the final product. When the former is not explicitly known ... simplicity merely serves as a rough indicator for the type of processing that took place prior to discovery.

Huber (St. Amant & Cohen, 1997; Huber, 1994) expresses thus the need for process-oriented evaluation:

Data analysis is different from, for example, word processing and batch programming: the correctness of the end product cannot be checked without inspecting the path leading to it.

Several pieces of previous work take into account the number of hypotheses being compared, and so can be considered early steps towards process-oriented evaluation. This includes notably systems that use the

Table 1: Empirical results: error rates and theory sizes of default CN2 and CN2 with process-oriented evaluation (CN2-POE).

Dataset	Error rate		Theory size	
	CN2	CN2-POE	CN2	CN2-POE
Breast	30.0±1.4	29.7±1.4	114.5±2.4	58.7±2.6
Echocardio	32.7±1.2	32.3±1.3	42.9±1.2	35.4±2.1
Glass	39.0±1.5	38.3±1.7	51.8±1.0	54.7±1.1
HeartC	20.8±0.8	22.5±0.8	57.8±0.9	52.0±1.0
HeartH	22.4±1.1	21.8±1.3	69.2±1.5	60.3±1.4
Hepatitis	21.2±0.9	19.2±1.3	40.2±1.7	34.0±1.3
Lympho	21.4±1.1	24.1±1.1	39.5±0.7	38.7±1.0
Soybean	19.5±1.0	19.4±1.0	116.7±2.3	110.9±3.1
Thyroid	4.1±0.2	3.8±0.2	97.5±2.0	104.8±2.0
Tumor	60.1±1.0	65.1±1.3	302.8±4.6	273.9±4.4
Voting	4.8±0.4	4.3±0.3	61.7±2.9	49.6±2.5

Bonferroni correction when testing significance (e.g., (Kass, 1980; Gaines, 1989; Jensen & Schmill, 1997); see also (Miller, 1981; Klockars & Sax, 1986; Westfall & Wolfinger, 1997)). A key difference between these systems and what is proposed here is that they require a somewhat arbitrary choice of significance threshold, while this paper directly attempts to optimize the end goal (expected generalization error). Also, the Bonferroni correction does not take hypothesis dependencies into account, while the present framework offers (at least in principle) a way of doing so.

Quinlan and Cameron-Jones's (1995) "layered search" method for automatically selecting CN2's beam width can also be considered a form of process-oriented evaluation. While layered search and CN2-POE have similar aims, their biases differ: layered search limits the search's width, while CN2-POE limits its length. The latter may be more effective in reducing the fragmentation and small disjuncts problems (Pagallo & Haussler, 1990; Holte, Acker & Porter, 1989). The assumptions made by the heuristic proposed here are also clearer than those implicit in Quinlan and Cameron-Jones's measure.

Evaluating models that are the result of a search process, not just of fitting the parameters of a pre-determined structure, has traditionally not been a concern of statisticians. However, this is beginning to change (Chatfield, 1995).

Some of the arguments made here for taking into account the number of hypotheses attempted are made in greater detail in (Cohen & Jensen, 1997) and (Ng, 1997). The present paper goes further in arguing that other aspects of the search process should also be taken

into account whenever possible (for example, in rule induction, the number of examples covered by each hypothesis).

6 FUTURE WORK

The development and evaluation contained in this paper are obviously only preliminary. As mentioned above, a version of CN2-POE that takes hypothesis dependencies into account is currently being implemented. Applications of POE to decision tree induction, backpropagation, instance selection, feature selection and discretization are also areas for future work. In each case, the main issue is likely to be finding the optimal trade-off between the computational and mathematical complexity of POE and its payoff in reduced error rates. The success of the enterprise is likely to hinge on distinguishing strong dependencies from weak ones that can be ignored, and on finding efficient but roughly correct approximations. For most learners in most domains, it is probably not realistic to expect large error reductions from POE, since it does not change the underlying representation or search process. However, if POE's gains are small but consistent across a broad spectrum of learners and domains, it will still be worth developing.

The POE error estimates introduced in this paper have two types of statistical bias. One stems from the fact that, because evaluation focuses on the lowest error found, low outliers have a stronger effect than high ones, leading to a negative bias (i.e., underestimating error). This bias can be estimated and the POE values corrected. This is an area of current work. The

second source of bias is the assumption that all hypotheses tried by the learner have similar error rates. This will lead to a positive bias when the error rate is decreasing (i.e., POE will tend to overestimate error at least up to the point where the learner starts overfitting). One way to overcome this is to introduce explicit expectations about the evolution of the learner's error as search progresses. For example, a specific type of curve may be assumed, or an "expected curve" can be compiled by cross-validation. Another approach is to avoid the assumption of similar error rates, for example by marginalizing over the true error rates of all hypotheses but the chosen one, or by using their maximum-likelihood estimates. Both of these approaches are also currently being studied.

The ultimate goal of POE is to accurately predict a hypothesis's generalization error from its training-set error, using knowledge of how the hypothesis was obtained. How far this is possible remains an open question.

7 CONCLUSION

Two main types of model selection are currently available. In *data-oriented evaluation*, a hypothesis's score does not depend on its form or how the hypothesis was found, but only on its performance on the data. In *representation-oriented evaluation*, the score depends on the data and on the hypothesis's form, but not on the search process that led to it. This paper argued that the latter cannot be ignored, and proposed *process-oriented evaluation* (POE), which takes all three factors into account. An application of POE to the CN2 rule induction system was found to reduce error in 8 of 11 benchmark datasets, and produce simpler theories in 9. Experiments in artificial domains support the hypothesis that these gains stem at least partly from CN2-POE's use of search process information.

References

- Akaike, H. (1978). A Bayesian analysis of the minimum AIC procedure. *Annals of the Institute of Statistical Mathematics*, 30A, 9–14.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.
- Brunk, C., & Pazzani, M. J. (1991). An investigation of noise-tolerant relational concept learning algorithms. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 389–393). Evanston, IL: Morgan Kaufmann.
- Chatfield, C. (1995). Model uncertainty, data mining and statistical inference. *Journal of the Royal Statistical Society A*, 158.
- Cheeseman, P. (1990). On finding the most probable model. In J. Shragar & P. Langley (Eds.), *Computational Models of Scientific Discovery and Theory Formation* (pp. 73–95). San Mateo, CA: Morgan Kaufmann.
- Cheeseman, P., & Oldford, R. W. (1994). Preface. In P. Cheeseman & R. W. Oldford (Eds.), *Selecting Models from Data: Artificial Intelligence and Statistics IV*. New York: Springer-Verlag.
- Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. *Proceedings of the Sixth European Working Session on Learning* (pp. 151–163). Porto, Portugal: Springer-Verlag.
- Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3, 261–283.
- Cohen, P. R., & Jensen, D. (1997). Overfitting explained. *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics* (pp. 115–122). Fort Lauderdale, FL: Society for Artificial Intelligence and Statistics.
- DeGroot, M. H. (1986). *Probability and Statistics* (2nd ed.). Reading, MA: Addison-Wesley.
- Domingos, P. (1997). Why does bagging work? A Bayesian account and its implications. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (pp. 155–158). Newport Beach, CA: AAAI Press.
- Domingos, P. (1998). Occam's two razors: The sharp and the blunt. Submitted.
- Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29, 103–130.
- Efron, B., & Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. New York: Chapman and Hall.
- Gaines, B. R. (1989). An ounce of knowledge is worth a ton of data. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 156–159). Ithaca, NY: Morgan Kaufmann.
- Heckerman, D. (1996). Bayesian networks for knowledge discovery. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 273–305). Menlo Park, CA: AAAI Press.

- Holte, R. C., Acker, L. E., & Porter, B. W. (1989). Concept learning and the problem of small disjuncts. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 813-818). Detroit, MI: Morgan Kaufmann.
- Huber, P. J. (1994). Languages for statistics and data analysis. In P. Dirschedl & R. Ostermann (Eds.), *Computational Statistics*. Heidelberg: Physica-Verlag.
- Jensen, D., & Schmill, M. (1997). Adjusting for multiple comparisons in decision tree pruning. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (pp. 195-198). Newport Beach, CA: AAAI Press.
- Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29, 119-127.
- Kearns, M. J., & Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. Cambridge, MA: MIT Press.
- Klockars, A. J., & Sax, G. (1986). *Multiple Comparisons*. Beverly Hills, CA: Sage.
- Lawrence, S., Giles, C. L., & Tsoi, A. C. (1997). Lessons in neural network training: Overfitting may be harder than expected. *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (pp. 540-545). Providence, RI: AAAI Press.
- MacKay, D. (1992). Bayesian interpolation. *Neural Computation*, 4, 415-447.
- Merz, C. J., Murphy, P. M., & Aha, D. W. (1997). UCI repository of machine learning databases. Department of Information and Computer Science, University of California at Irvine, Irvine, CA.
- Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20, 111-161.
- Miller, Jr., R. G. (1981). *Simultaneous Statistical Inference* (2nd ed.). New York: Springer-Verlag.
- Moody, J. E. (1992). The effective number of parameters: An analysis of generalization and regularization in learning systems. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4* (pp. 847-854). San Mateo, CA: Morgan Kaufmann.
- Murphy, P., & Pazzani, M. (1994). Exploring the decision forest. *Journal of Artificial Intelligence Research*, 1, 257-275.
- Ng, A. Y. (1997). Preventing "overfitting" of cross-validation data. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 245-253). Nashville, TN: Morgan Kaufmann.
- Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 3, 71-99.
- Pearl, J. (1978). On the connection between the complexity and credibility of inferred models. *International Journal of General Systems*, 4, 255-264.
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C* (2nd ed.). Cambridge, UK: Cambridge University Press.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- Quinlan, J. R., & Cameron-Jones, R. M. (1995). Oversearching and layered search in empirical learning. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1019-1024). Montréal, Canada: Morgan Kaufmann.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14, 465-471.
- Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, 10, 153-178.
- Schuermans, D., Ungar, L. H., & Foster, D. P. (1997). Characterizing the generalization performance of model selection strategies. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 340-348). Nashville, TN: Morgan Kaufmann.
- Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6, 461-464.
- St. Amant, R., & Cohen, P. R. (1997). Building an EDA assistant: A progress report. *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics* (pp. 501-512). Ft. Lauderdale, FL: Society for Artificial Intelligence and Statistics.
- Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36, 111-147.
- Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. New York: Springer-Verlag.
- Wallace, C. S., & Boulton, D. M. (1968). An information measure for classification. *Computer Journal*, 11, 185-194.
- Webb, G. I. (1996). Further experimental evidence against the utility of Occam's razor. *Journal of Artificial Intelligence Research*, 4, 397-417.
- Westfall, P. H., & Wolfinger, R. D. (1997). Multiple tests with discrete distributions. *American Statistician*, 51, 3-8.

Relational Reinforcement Learning

Sašo Džeroski

Department of Intelligent Systems
Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si

Luc De Raedt, Hendrik Blockeel

Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
{Luc.DeRaedt, Hendrik.Blockeel}@cs.kuleuven.ac.be

Abstract

Relational reinforcement learning is presented, a learning technique that combines reinforcement learning with relational learning or inductive logic programming. Due to the use of a more expressive representation language to represent states, actions and Q-functions, relational reinforcement learning can be potentially applied to a new range of learning tasks. One such task that we investigate is planning in the blocks world, where it is assumed that the effects of the actions are unknown to the agent and the agent has to learn a policy. Within this simple domain we show that relational reinforcement learning solves some existing problems with reinforcement learning. In particular, relational reinforcement learning allows us to employ structural representations, make abstraction of specific goals pursued and exploit the results of previous learning phases when addressing new (more complex) situations.

1 INTRODUCTION

Within the field of machine learning, both reinforcement learning [8] and inductive logic programming (or relational learning) [12, 10] have received a lot of attention since the early nineties. It is therefore no surprise that both Leslie Pack Kaelbling and Richard Sutton (in their invited talks at IJCAI-97, Nagoya, Japan) suggested to study the combination of these two fields.

From the reinforcement learning point of view, this could significantly extend the application perspective. Most representations used in reinforcement learning are inadequate for describing planning tasks such as the simple blocks world. Even reinforcement learning

work that involves generalization has largely employed an attribute-value representation. Furthermore, due to the use of variables in relational representations, it is possible to make abstractions of some specific details of the learning tasks, such as the goal pursued. Indeed, when learning to plan in the blocks world, one would expect that the results of learning how to stack block *a* onto block *b* would be similar to stacking *c* onto *d*. Current approaches to reinforcement learning have to retrain from scratch if the goal is changed in this manner. Using relational reinforcement learning retraining is unnecessary. Relational reinforcement learning also allows us to exploit the results of learning in a simple domain when learning in a more complex domain (e.g., going from 3 blocks to 4 blocks in the blocks world).

From the inductive logic programming point of view, it is important to address domains such as reinforcement learning. So far, inductive logic programming has mainly studied concept-learning, and largely ignored the rest of machine learning. By demonstrating the potential of relational representations for reinforcement learning, we hope to show that the relational learning methodology does not only apply to concept-learning but to the whole field of machine learning.

With this in mind, we present a preliminary approach to relational reinforcement learning and apply it to simple planning tasks in the blocks world. The planning task involves learning a policy to select actions. Learning is necessary as the planning agent does not know the effects of its actions. Relational reinforcement learning employs the Q-learning method [14, 8, 11] where the Q-function is learned using a relational regression tree algorithm (see [6, 9]). A state is represented relationally as a set of ground facts. A relational regression tree in this context takes as input a relational description of a state, a goal and an action, and produces the corresponding Q-value.

This paper is organized as follows. In section 2, we view planning (under uncertainty) as a reinforcement learning task, and in section 3, we briefly review reinforcement and in particular Q-learning. Section 4 introduces relational reinforcement learning that combines Q-learning and logical regression trees. In section 5, we present some experiments, and finally, in section 6, we conclude and touch upon related work.

2 LEARNING TO PLAN AS REINFORCEMENT LEARNING

Consider a planning agent with the following task:

Given

- a set of possible states S ,
- a set of possible actions A ,
- an UNKNOWN function $\delta: S \times A \rightarrow A$,
- a function $pre: S \times A \rightarrow \{t, f\}$,
- a goal $goal: S \rightarrow \{t, f\}$, and
- a starting state $s \in S$,

find a sequence of actions a_1, \dots, a_n ($a_i \in A$) such that

- $goal(\delta(\dots\delta(s, a_1))\dots, a_n) = t$, and
- $pre(\delta(\dots\delta(s, a_1))\dots, a_i) = t$.

The agent can be in one of the states of S . It can execute action $a \in A$ in a given state s if the preconditions for a are true in s ($pre(s, a) = t$), e.g., as in STRIPS [7]. Executing an action a in a state s will put the agent in a new state $\delta(s, a)$. When placed in a state s the task of the agent is to find a (shortest) sequence of actions a_1, \dots, a_n that will lead it to a goal state. The prototypical AI task belonging to this category is planning.

It is assumed here that the agent does not know the effect of its actions, hence the function δ is unknown to the agent. The above task specification thus contrasts with classical planning in that the δ function is unknown to the agent. Therefore, this task requires a learning component.

Example: The best known (toy)-domain to study planning is the blocks world. Consider the situation where we have three blocks called a , b and

c , and the floor. Blocks can be on the floor or can be stacked on each other. Each state can be described by a set (list) of facts, e.g., $s_1 = \{clear(a), on(a, b), on(b, c), on(c, floor)\}$. The available actions are then $move(x, y)$ where $x \neq y$ and $x \in \{a, b, c\}$, $y \in \{a, b, c, floor\}$.

It is then possible to define the preconditions and effects of actions. The Prolog code below defines pre and δ respectively. The predicate pre defines the preconditions for the action $move(X, Y)$ while the predicate $delta$ defines its effects: $delta(S, A, S1)$ succeeds when $\delta(S, A) = S1$. States are represented as lists of facts and the auxiliary predicate $holds(S, Query)$ succeeds when $Query$ would succeed in the knowledge base containing the facts in S only.

```
pre(S, move(X, Y)) :-
    holds(S, [clear(X), clear(Y),
              not X=Y, not on(X, floor)]).
pre(S, move(X, Y)) :-
    holds(S, [clear(X), clear(Y),
              not X=Y, on(X, floor)]).
pre(S, move(X, floor)) :-
    holds(S, [clear(X), not on(X, floor)]).

holds(S, []).
holds(S, [not X=Y | R]) :-
    not X=Y, !, holds(S, R).
holds(S, [not A | R]) :-
    not member(A, S), holds(S, R).
holds(S, [A | R]) :-
    member(A, R), holds(S, R).

delta(S, move(X, Y), NextS) :-
    holds(S, [clear(X), clear(Y),
              not X=Y, not on(X, floor)]),
    delete([clear(Y), on(X, Z)], S, S1),
    add([clear(Z), on(X, Y)], S1, NextS).
delta(S, move(X, Y), NextS) :-
    holds(S, [clear(X), clear(Y),
              not X=Y, on(X, floor)]),
    delete([clear(Y), on(X, floor)], S, S1),
    add([on(X, Y)], S1, NextS).
delta(S, move(X, floor), NextS) :-
    holds(S, [clear(X), not on(X, floor)]),
    delete([on(X, Z)], S, S1),
    add([clear(Z), on(X, floor)], S1, NextS).
```

The goal is to stack a onto b , i.e.,
 $goal(S) :- member(on(a, b), S).$ □

3 REINFORCEMENT LEARNING

Planning with incomplete knowledge as outlined above can be recast as a reinforcement learning problem.

3.1 THE BASICS OF REINFORCEMENT LEARNING

The basic notions of reinforcement learning can be outlined as follows (we follow the notation used by Mitchell [11]).

- The task of the agent is to learn a policy $\pi : S \rightarrow A$ for selecting its next action a_t based on the current state s_t ; that is $\pi(s_t) = a_t$.
- The reward at time t is $r_t = r(s_t, a_t)$. We will assume here that $r_t = 1$ if $goal(\delta(s_t, a_t)) = t$ and $s_t \neq \delta(s_t, a_t)$; otherwise $r_t = 0$. The reward function r is unknown to the learner as it relies on the unknown δ . The reward function only gives a reward in goal states.
- The state at time $t + 1$ is $s_{t+1} = \delta(s_t, a_t)$ if $goal(s_t) = f$; otherwise $s_{t+1} = s_t$. This captures the idea that goal states are absorbing states, i.e., once a goal state is reached the only available action is to stay in the state.
- The learned policy should be optimal, i.e., it should maximize

$$V^\pi(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$

where $0 \leq \gamma < 1$. We will denote the optimal policy by π^* .

The optimal policy π^* allows us to compute the shortest plan to reach a goal state. So, learning the optimal policy (or approximations thereof) will allow us to improve our planning performance.

3.2 Q-LEARNING

It is well-known that under the conditions sketched in the previous subsection, Q-learning allows us to approximate the optimal policy.

The optimal policy π^* will always select the action that maximizes the sum of the immediate reward and the value of the immediate successor state, i.e.,

$$\pi^*(s) = \operatorname{argmax}_a (r(s, a) + \gamma V^{\pi^*}(\delta(s, a)))$$

The problem with this formulation of π^* is that it requires knowledge of δ and r , which the learner does not have at its disposal.

The Q-function is defined as follows :

$$Q(s, a) = r(s, a) + \gamma V^{\pi^*}(\delta(s, a))$$

Knowing Q allows us to rewrite the definition of π^* as follows :

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

According to Mitchell, this rewrite is important as it shows that if the agent can learn the Q function instead of the V^{π^*} function, it will be able to act optimally. The Q -function for a fixed *goal* can then be approximated by \hat{Q} , for which a look-up table is learned by the following algorithm (cf. [11]).

```

for each  $s, a$  do
  initialize the table entry  $\hat{Q}(s, a) = 0$ 
do forever
   $i := 0$ 
  generate a random state  $s_0$ 
  while not  $goal(s_i)$  do
    select an action  $a_i$  and execute it
    receive an immediate reward  $r_i = r(s_i, a_i)$ 
    observe the new state  $s_{i+1}$ 
     $i := i + 1$ 
  for  $j = i - 1$  to 0 do
    update  $\hat{Q}(s_j, a_j) := r_i + \gamma \max_{a'} \hat{Q}(s_{j+1}, a')$ 

```

It is common in Q-learning to select action a in state s probabilistically so that $P(a|s)$ is proportional to $\hat{Q}(s, a)$, e.g.,

$$P(a_i|s) = k^{\hat{Q}(s, a_i)} / \sum_j k^{\hat{Q}(s, a_j)} \quad (1)$$

Higher values of k give stronger preference to actions with high values of \hat{Q} causing the agent to exploit what it has learned, while lower values of k reduce this preference allowing the agent to explore actions that currently do not have high values of \hat{Q} .

4 RELATIONAL REINFORCEMENT LEARNING

4.1 THE NEED FOR RELATIONAL REPRESENTATIONS

Given the above classical framework for Q-learning we could now learn to plan in the blocks world sketched earlier. Using the approach as it stands we could store all the state-action pairs encountered and memorize/update the corresponding Q values, having in effect an explicit look-up table for state-action pairs. This has however a number of disadvantages:

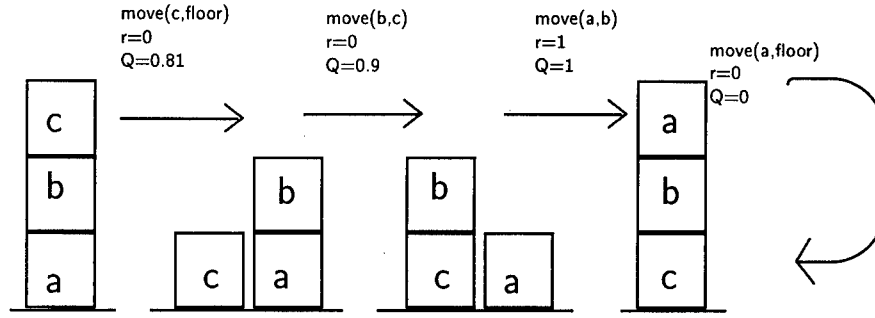


Figure 1: A blocks-world example for relational Q-learning.

- It is impractical for all but the smallest state-spaces. Furthermore, using look-up tables does not work for infinite state spaces which could arise when first order representations are used (e.g., if the number of blocks in the world is unknown or infinite the above method does not work).
- Despite the use of a relational representation for states and actions, the above method is unable to capture the structural aspects of the planning task.
- Whenever the goal is changed from say $on(a, b)$ to $on(b, c)$ the above method would require retraining the whole Q function.
- Ideally, one would expect that the results of learning in a world with 3 blocks could be (partly) recycled when learning in a 4 blocks world later on. It is unclear how to achieve this with the lookup table.

The first problem can be solved by using an inductive learning algorithm (e.g., a neural network) to approximate \hat{Q} . The three other problems can only be solved by using a *relational* learning algorithm that can make abstraction of the specific blocks and goals using variables. We now present such a relational learning algorithm.

4.2 THE RRL ALGORITHM

The relational reinforcement learning (RRL) algorithm is obtained by combining the classical Q-learning algorithm with stochastic selection of actions and a relational regression algorithm. Instead of having an explicit lookup table, an implicit representation of the Q-function is learned in the form of a logical regression tree, called a Q-tree.

The main point where RRL differs from the algorithm in section 3.2 is in the for-loop where the \hat{Q} function is modified. This for-loop now becomes :

```

for j=i-1 to 0 do
    generate example  $(s_j, a_j, \hat{q}_j)$ ,
    where  $\hat{q}_j := r_i + \gamma \max_{a'} \hat{Q}_e(s_{j+1}, a')$ 
    update  $\hat{Q}_e$  using TILDE-RT
    to produce  $\hat{Q}_{e+1}$  using the examples  $(s_j, a_j, \hat{q}_j)$ 
    
```

TILDE-RT [6] is an algorithm for learning logical regression trees and will be described briefly below.

The initial tree \hat{Q}_0 assigns zero value to all state-action pairs. From each goal state g encountered, an example $(g, a, 0)$ is generated for each action a whose preconditions are satisfied in g . The rationale for this is that no reward can be expected from applying an action in an absorbing goal state.

Example: A possible initial episode ($e = 0$) in the blocks world with three blocks a , b , and c , where the goal is to stack a on b (i.e., $goal(on(a, b))$) is depicted in Figure 1. The discount factor γ is 0.9 and the reward given is one on achieving a goal state, zero otherwise.

The examples generated by RRL use the actions and the Q-values listed above the arrows representing the actions. The actual format of these examples is listed in Table 1. It is exactly this input that would be used by TILDE-RT to generate the Q-tree Q_1 . \square

TILDE-RT is not incremental, so we currently simulate the update of \hat{Q} by keeping all (s, a) pairs encountered and the most recent \hat{q} value for each pair, and inducing a relational regression tree \hat{Q}_e from these examples after each episode e . This tree is then used to select actions in episode $e + 1$.

Table 1: Examples for TILDE-RT generated from the blocks-world Q-learning episode in Figure 1.

qvalue(0.81). action(move(c,floor)). goal(on(a,b)). clear(c). on(c,b). on(b,a). on(a,floor).	qvalue(0.9). action(move(b,c)). goal(on(a,b)). clear(b). clear(c). on(b,a). on(a,floor). on(c,floor).	qvalue(1.0). action(move(a,b)). goal(on(a,b)). clear(a). clear(b). on(b,c). on(a,floor). on(c,floor).	qvalue(0.0). action(move(a,floor)). goal(on(a,b)). clear(a). on(a,b). on(b,c). on(c,floor).
--	--	--	---

4.3 TOP-DOWN INDUCTION OF LOGICAL REGRESSION TREES

Logical regression trees are similar to propositional regression trees [3]: leaves predict a value for a continuous class, while internal nodes contain conditions that partition the example space. The difference is that examples here are not feature or attribute-value vectors, but sets of relational facts, representing, e.g., a state of the blocks world, a goal, and an action to be taken, all at the same time. Similarly, internal nodes are not restricted to attribute-value tests but can be first order literals containing predicates, variables and complex terms.

The TILDE-RT system [6] induces such first order logical regression trees (or relational regression trees) from examples (cf. [9] for a related approach). The input for TILDE-RT is a set of state-action pairs together with the corresponding Q-values, represented as sets of facts. From this TILDE-RT induces (using the classical TDIDT-algorithm) a tree in which the classes correspond to real numbers (Q-values).

To illustrate the above notions, consider the episode shown in Figure 1. The examples for TILDE-RT generated by the RRL algorithm are given in Table 1. The relational regression tree induced by TILDE-RT from these examples is shown in Figure 2.

Nodes in the tree correspond to Prolog-queries. If the query succeeds in an example the *yes* subtree is taken, otherwise the *no* subtree. Different nodes in the tree may share variables, e.g., the bottom node in the tree (containing `action(move(D,B))`) refers to the variable *D* that first appear in the root of the tree (`goal(on(C,D))`). The Prolog program corresponding to the tree is shown in the lower part of Figure 2.

The semantics of logical decision trees is extensively discussed in [1], as well as the correspondence between a tree and a Prolog program. The method to induce the trees is described in [6] and is - for the case of regression trees - very similar to Kramer's SRT system [9]. We refer to these papers for more details on the representation and learning of such trees.

To find the Q-value corresponding to a state-action pair, one has to construct a Prolog knowledge base containing the Prolog program (corresponding to the tree), all facts in the state, the action, and the goal. Running the query `?-qvalue(Q)` will then return the desired result. E.g., the Q-tree above will return a Q-value of zero for all actions if the goal is `on(C,D)` and `on(C,D)` holds in the state (goal states are absorbing). On the other hand, if the goal `on(C,D)` does not yet hold and the action is `move(C,D)`, then a Q-value of one is returned (reward of one for achieving a goal state).

```

action(move(A,B)) , goal(on(C,D))
on(C,D) ?
---yes: [0]
---no:  action(move(C,D)) ?
        ---yes: [1]
        ---no:  action(move(D,B)) ?
                ---yes: [0.9]
                ---no:  [0.81]

```

```

qvalue(0) :-
    action(move(A,B)) , goal(on(C,D)) ,
    on(C,D), !.
qvalue(1) :-
    action(move(A,B)) , goal(on(C,D)) ,
    action(move(C,D)), !.
qvalue(0.9) :-
    action(move(A,B)) , goal(on(C,D)) ,
    action(move(D,B)), !.
qvalue(0.81).

```

Figure 2: A relational regression tree generated by TILDE-RT from the examples in Table 1 and its equivalent Prolog program.

```

action(move(A,B)) , goal(on(C,D))
on(C,D) ?
+--yes: [0]
+--no: action(move(C,D)) ?
      +--yes: [1]
      +--no: on(B,C) ?
            +--yes: [0.729]
            +--no: on(B,D) ?
                  +--yes: [0.729]
                  +--no: action(move(A,C)) ?
                        +--yes: [0.81]
                        +--no: action(move(A,D)) ?
                              +--yes: [0.81]
                              +--no: clear(D) ?
                                    +--yes: on(C,B) ?
                                          | +--yes: on(A,C) ?
                                          | | +--yes: [0.9]
                                          | | +--no: clear(C) ?
                                          | | +--yes: [0.9]
                                          | | +--no: [0.81]
                                          | +--no: [0.9]
                                    +--no: clear(C) ?
                                          +--yes: on(C,B) ?
                                          | +--yes: [0.9]
                                          | +--no: [0.81]
                                          +--no: [0.81]
    
```

Figure 3: The Q-tree generated by RRL in the 3 blocks world after 10 episodes.

5 EXPERIMENTS

We applied the RRL algorithm described above to learn how to stack one block onto another in worlds with three and four blocks, respectively. In particular, the goal to achieve was $on(a, b)$, the two other blocks being c and d . An example episode in the three blocks world is depicted in Figure 1.

The discount factor γ had the value 0.9. When selecting states stochastically according to equation 1, the constant k was set to $e^{0.2}$. Examples for learning Q-trees were generated after each episode, as described in the section above.

TILDE-RT was used to induce an updated Q-tree after each episode. The minimal number of cases in a leaf was set to one and TILDE-RT generated unpruned trees, which exactly reproduce the Q-values for the state-action pairs seen during the learning phase.

Using the above settings, the RRL algorithm was first run for 10 episodes in the 3 blocks world. The tree shown in Figure 3 was generated by TILDE-RT after the final episode. This tree represents the optimal policy for the given reinforcement learning problem. The top two levels of the tree match those of the tree in Table 1, which was generated from a single episode.

It is important to note that the individual blocks are not referred to in the tree itself directly, but only through the variables of the goal. This means that the tree represents the optimal policy not only for achieving the goal $on(a, b)$, but also $on(b, c)$ and $on(c, a)$. This is one of the major advantages of using a relation representation for Q-learning.

The Q-tree obtained after 10 episodes in the 4 blocks worlds was much larger (44 nodes as opposed to the 12 nodes of the 3-blocks Q-tree). It also represents an optimal policy: it chooses a shortest path to a goal state from all initial states, if the action with the highest Q-value is always selected.

The 3 top levels of the tree match with the tree from the 3 blocks world. This indicates that the result of learning in the 3 blocks world could be used to bootstrap learning in the 4 blocks world. Indeed, if we take the Q-tree learned in the 3 blocks world shown in Figure 3 and use it to select actions in the 4 blocks world, it selects an optimal path to a goal state from all but 9 of the 73 possible initial states. In 4 of the 9 cases a looping behavior is produced, in the remaining 5 cases one extra action is needed as compared to an optimal plan.

Using the Q-tree from Figure 3 to bootstrap RRL in the 4 blocks world helps improve performance, especially in the initial episodes. Without bootstrapping, after two episodes a tree is learned which produces nonoptimal behavior in 12 of the 73 initial states. With bootstrapping, the behavior of the learned tree is nonoptimal for 8 of the 73 possible initial states. After ten episodes, the learned Q-tree produces optimal behavior and is much smaller (27 nodes) as compared to the Q-tree learned without bootstrapping (44 nodes).

6 DISCUSSION

We have presented an approach to planning with incomplete knowledge that combines reinforcement learning and relational regression into a technique called relational reinforcement learning. The advantages of this approach include the ability to use structured representations, which enables us to also describe infinite worlds, and the ability to use variables, which allows us to abstract away from specific details of the situations (such as, e.g., the goal). The ability to use results of simpler tasks to bootstrap learning in more complex tasks is also an advantage worth mentioning. Finally, it is easy to incorporate nondeterministic actions within the proposed approach.

Even for standard reinforcement learning, scaling-up as the dimensionality of the problem increases can be a problem. Using a richer description language may seem to make things even worse. However, there are reasons to expect that using a richer representation actually enables relational Q-learning to scale-up better than standard Q-learning. Let us illustrate these on the blocks world.

First, in the representation employed, the relational theories learned abstract away the block names, causing the number of states that are essentially different to decrease. For instance, with $goal(on(a, b))$ the states $\{on(a, c), on(c, b), on(b, floor), on(d, floor)\}$ and $\{on(a, d), on(d, b), on(b, floor), on(c, floor)\}$ are

essentially the same as c and d are interchangeable. In standard Q-learning, they would be considered different. In our 4-blocks example, the number of states that essentially differ from one another is 73 for a standard Q-learner, but only 38 for a relational one. This ratio increases combinatorially (since all blocks that do not occur in the goal have no special status and are thus interchangeable, the ratio increases roughly with $(n - 2)!$, where n is the total number of blocks).

Second, the use of background knowledge makes it possible to abstract even further from specific situations that do not essentially differ. For instance, when a has to be cleared in order to be able to move it, it is not essential whether there are 1, 5 or 17 blocks above a : the top of the stack on a should be moved. Using background definitions such as $above(X, Y)$ (the recursive closure of $on(X, Y)$) it is possible to state a rule such as "if there are blocks on a , move the topmost of those blocks to the floor" which captures a very large set of specific cases.

However, the exact scale-up behavior of relational reinforcement learning has still to be determined experimentally. The experimental evaluation of our approach done so far is preliminary and is mainly intended to highlight the principal advantages of using a relational representation for reinforcement learning. We hope that this paper will inspire further research into the combination of relational and reinforcement learning, as much work remains to be done. This includes work in the line of proper performance assessment, both in terms of standard performance tests in reinforcement learning fashion (root mean square errors of learned Q-values wrt. the Q-values of the optimal policy) and in considering more complex and demanding planning problems.

More complex problems can be obtained by increasing the number of blocks in the world, considering more complex goals, such as building a stack of all available blocks, and considering problems outside the blocks world.

This work is related to work on generalization in reinforcement learning, which has however mainly addressed the use of neural networks for this purpose [13]. The closest related work is probably Chapman's and Kaelbling's decision tree algorithm that was specifically designed for reinforcement learning [5]. Note however that our approach is distinguished from the mainstream work in reinforcement learning by the use of a relational representation.

Relational representations are commonly used in planning approaches. There have also been some attempts to combine planning with relational learning within those approaches, e.g., within the PRODIGY approach [2]. Our approach is related to them through the use of a relational representation. However, it seems that the combination of planning, reinforcement learning and relational learning has not been addressed so far.

The reinforcement learning part of the work presented in this paper is admittedly simple. We have taken a standard textbook description of reinforcement learning [11] and incorporated an implementation of it within our approach. We have considered a deterministic setting and a goal-oriented formulation of the learning problem. However, both restrictions can be easily lifted to extend to non-zero rewards on non-terminal states (the RRL algorithm actually makes no assumption on the reinforcement received) and non-deterministic actions. To handle nondeterministic actions an appropriate update rule (see page 382 of [11]) has to be used to generate examples for the TILDE-RT algorithm. Other points where the reinforcement learning part can be improved include the initialization of Q values and the exploration strategy.

The current implementation of TILDE-RT is - according to reinforcement standards - not optimal. One of the reasons is that it is not incremental. However, incrementality is not enough, as the (estimated) values of Q are changing with time. These problems are taken care of within the Chapman and Kaelbling's decision tree algorithm that was specifically designed for reinforcement learning [5]. A natural direction for further work is thus to develop a first order regression tree algorithm combining the representations of TILDE-RT with the algorithm and performance measures of the approach by Chapman and Kaelbling. Such an integrated approach, which is currently under development, would not suffer from the abovementioned problems.

Acknowledgements

This work was supported in part by the ESPRIT IV Project 20237 ILP2. Luc De Raedt is supported by the Fund for Scientific Research of Flanders. Hendrik Blockeel is supported by the Flemish Institute for the Promotion of Scientific and Technological Research in Industry (IWT).

References

- [1] Blockeel, H., and De Raedt, L. (1997) Experiments with Top-down Induction of Logical Decision Trees. *Artificial Intelligence*. Forthcoming.
- [2] Borrajo, D., and Veloso, M. (1997) Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review*, 11(1-5): 371-405.
- [3] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth, Belmont.
- [4] Blockeel, H., and De Raedt, L. (1997) Lookahead and discretization in ILP. In *Proc. 7th Intl. Workshop on Inductive Logic Programming*, pages 77-84, Springer, Berlin.
- [5] Chapman, D., and Kaelbling, L. (1991) Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proc. 12th Intl. Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.
- [6] De Raedt, L., and Blockeel, H. (1997) Using logical decision trees for clustering. In *Proc. 7th Intl. Workshop on Inductive Logic Programming*, pages 133-141, Springer, Berlin.
- [7] Fikes, R.E., and Nilsson, N.J. (1971) STRIPS: A new approach to the application of theorem proving. *Artificial Intelligence*, 2(3/4): 189-208.
- [8] Kaelbling, L., Littman, M., and Moore, A. (1996) Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4: 237-285.
- [9] Kramer, S. (1996) Structural regression trees. In *Proc. 13th Natl. Conf. on Artificial Intelligence*. AAAI Press, Menlo Park, CA.
- [10] Lavrač, N. and Džeroski, S. (1994) *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester.
- [11] Mitchell, T. (1997) *Machine Learning*. McGraw-Hill, New York.
- [12] Muggleton, S., and De Raedt, L. (1994) Inductive logic programming: Theory and methods. *Journal of Logic Programming* 19/20: 629-679.
- [13] Tesauro, G. (1995) Temporal difference learning and TD-GAMMON. *Communications of the ACM*, 38(3): 58-68.
- [14] Watkins, C., and Dayan, P. (1992) Q-learning. *Machine Learning*, 8: 279-292.

Generating Accurate Rule Sets Without Global Optimization

Eibe Frank

Department of Computer Science
University of Waikato
Hamilton, New Zealand
eibe@cs.waikato.ac.nz

Ian H. Witten

Department of Computer Science
University of Waikato
Hamilton, New Zealand
ihw@cs.waikato.ac.nz

Abstract

The two dominant schemes for rule-learning, C4.5 and RIPPER, both operate in two stages. First they induce an initial rule set and then they refine it using a rather complex optimization stage that discards (C4.5) or adjusts (RIPPER) individual rules to make them work better together. In contrast, this paper shows how good rule sets can be learned one rule at a time, without any need for global optimization. We present an algorithm for inferring rules by repeatedly generating partial decision trees, thus combining the two major paradigms for rule generation—creating rules from decision trees and the separate-and-conquer rule-learning technique. The algorithm is straightforward and elegant: despite this, experiments on standard datasets show that it produces rule sets that are as accurate as and of similar size to those generated by C4.5, and more accurate than RIPPER's. Moreover, it operates efficiently, and because it avoids postprocessing, does not suffer the extremely slow performance on pathological example sets for which the C4.5 method has been criticized.

1 Introduction

If-then rules are the basis for some of the most popular concept description languages used in machine learning. They allow “knowledge” extracted from a dataset to be represented in a form that is easy for people to understand. This gives domain experts the chance to analyze and validate that knowledge, and combine it

with previously known facts about the domain.

A variety of approaches to learning rules have been investigated. One is to begin by generating a decision tree, then to transform it into a rule set, and finally to simplify the rules (Quinlan, 1987a); the resulting rule set is often more accurate than the original tree. Another is to use the “separate-and-conquer” strategy (Pagallo & Haussler, 1990) first applied in the AQ family of algorithms (Michalski, 1969) and subsequently used as the basis of many rule learning systems (Fürnkranz, 1996). In essence, this strategy determines the most powerful rule that underlies the dataset, separates out those examples that are covered by it, and repeats the procedure on the remaining examples.

Two dominant practical implementations of rule-learners have emerged from these strands of research: C4.5 (Quinlan, 1993) and RIPPER (Cohen, 1995). Both perform a global optimization process on the set of rules that is induced initially. The motivation for this in C4.5 is that the initial rule set, being generated from a decision tree, is unduly large and redundant: C4.5 drops some individual rules (having previously optimized rules locally by dropping conditions from them). The motivation in RIPPER, on the other hand, is to increase the accuracy of the rule set by replacing or revising individual rules. In either case the two-stage nature of the algorithm remains: as Cohen (1995) puts it, “... both RIPPER and C4.5 rules start with an initial model and iteratively improve it using heuristic techniques.” Experiments show that both the size and the performance of rule sets are significantly improved by post-induction optimization. On the other hand, the process itself is rather complex and heuristic.

This paper presents a rule-induction procedure that avoids global optimization but nevertheless produces

accurate, compact rule sets. The method combines the two rule learning paradigms identified above. Section 2 discusses these two paradigms and their incarnation in C4.5 and RIPPER. Section 3 presents the new algorithm, which we call "PART" because it is based on partial decision trees. Section 4 describes an experimental evaluation on standard datasets comparing PART to C4.5, RIPPER, and C5.0, the commercial successor of C4.5.¹ Section 5 summarizes our findings.

2 Related Work

We review two basic strategies for producing rule sets. The first is to begin by creating a decision tree and then transform it into a rule set by generating one rule for each path from the root to a leaf. Most rule sets derived in this way can be simplified dramatically without losing predictive accuracy. They are unnecessarily complex because the disjunctions that they imply can often not be expressed succinctly in a decision tree. This is sometimes known as the "replicated subtree" problem (Pagallo & Haussler, 1990).

When obtaining a rule set, C4.5 first transforms an unpruned decision tree into a set of rules in the aforementioned way. Then each rule is simplified separately by greedily deleting conditions in order to minimize the rule's estimated error rate. Following that, the rules for each class in turn are considered and a "good" subset is sought, guided by a criterion based on the minimum description length principle (Rissanen, 1978) (this is performed greedily, replacing an earlier method that used simulated annealing). The next step ranks the subsets for the different classes with respect to each other to avoid conflicts, and determines a default class. Finally, rules are greedily deleted from the whole rule set one by one, so long as this decreases the rule set's error on the training data.

The whole process is complex and time-consuming. Five separate stages are required to produce the final rule set. It has been shown that for noisy datasets, runtime is cubic in the number of instances (Cohen, 1995). Moreover, despite the lengthy optimization process, rules are still restricted to conjunctions of those attribute-value tests that occur along a path in the initial decision tree.

Separate-and-conquer algorithms represent a more direct approach to learning decision rules. They generate one rule at a time, remove the instances cov-

ered by that rule, and iteratively induce further rules for the remaining instances. In a multi-class setting, this automatically leads to an ordered list of rules, a type of classifier that has been termed a "decision list" (Rivest, 1987). Various different pruning methods for separate-and-conquer algorithms have been investigated by Fürnkranz (1997), who shows that the most effective scheme is to prune each rule back immediately after it is generated, using a separate stopping criterion to determine when to cease adding rules (Fürnkranz & Widmer, 1994). Although originally formulated for two-class problems, this procedure can be applied directly to multi-class settings by building rules separately for each class and ordering them appropriately (Cohen, 1995).

RIPPER implements this strategy using reduced error pruning (Quinlan, 1987b), which sets some training data aside to determine when to drop the tail of a rule, and incorporates a heuristic based on the minimum description length principle as stopping criterion. It follows rule induction with a post-processing step that revises the rule set to more closely approximate what would have been obtained by a more expensive global pruning strategy. To do this, it considers "replacing" or "revising" individual rules, guided by the error of the modified rule set on the pruning data. It then decides whether to leave the original rule alone or substitute its replacement or revision, a decision that is made according to the minimum description length heuristic. It has been claimed (Cohen, 1995) that RIPPER generates rule sets that are as accurate as C4.5's. However, our experiments on a large collection of standard datasets—reported in Section 3—do not confirm this.

As the following example shows, the basic strategy of building a single rule and pruning it back can lead to a particularly problematic form of overpruning, which we call "hasty generalization." This is because the pruning interacts with the covering heuristic. Generalizations are made before their implications are known, and the covering heuristic then prevents the learning algorithm from discovering the implications.

Here is a simple example of hasty generalization. Consider a Boolean dataset with attributes *a* and *b* built from the three rules in Figure 1, corrupted by ten percent class noise. Assume that the pruning operator is conservative and can only delete a single final conjunction of a rule at a time (not an entire tail of conjunctions as RIPPER does). Assume further that the first

¹A test version of C5.0 is available from <http://www.rulequest.com>.

Rule	Coverage			
	Training Set		Pruning Set	
	\oplus	\ominus	\oplus	\ominus
1: $a = \text{true} \Rightarrow \oplus$	90	8	30	5
2: $a = \text{false} \wedge b = \text{true} \Rightarrow \oplus$	200	18	66	6
3: $a = \text{false} \wedge b = \text{false} \Rightarrow \ominus$	1	10	0	3

Figure 1: A hypothetical target concept for a noisy domain.

rule has been generated and pruned back to

$$a = \text{true} \Rightarrow \oplus$$

(The training data in Figure 1 is solely to make this scenario plausible.) Now consider whether the rule should be further pruned. Its error rate on the pruning set is 5/35, and the null rule

$$\Rightarrow \oplus$$

has an error rate of 14/110, which is smaller. Thus the rule set will be pruned back to this single, trivial, rule, instead of the patently more accurate three-rule set shown in Figure 1.

Hasty generalization is not just an artifact of reduced error pruning: it can happen with pessimistic pruning (Quinlan, 1993) too. Because of variation in the number of noisy instances in the data sample, one can always construct situations in which pruning causes rules with comparatively large coverage to swallow rules with smaller (but still significant) coverage. This can happen whenever the number of errors committed by a rule is large compared with the total number of instances covered by an adjacent rule.

3 Obtaining Rules From Partial Decision Trees

The new method for rule induction, PART, combines the two approaches discussed in Section 2 in an attempt to avoid their respective problems. Unlike both C4.5 and RIPPER it does not need to perform global optimization to produce accurate rule sets, and this added simplicity is its main advantage. It adopts the separate-and-conquer strategy in that it builds a rule, removes the instances it covers, and continues creating rules recursively for the remaining instances until none are left. It differs from the standard approach

in the way that each rule is created. In essence, to make a single rule a pruned decision tree is built for the current set of instances, the leaf with the largest coverage is made into a rule, and the tree is discarded. This avoids hasty generalization by only generalizing once the implications are known (i.e., all the subtrees have been expanded).

The prospect of repeatedly building decision trees only to discard most of them is not as bizarre as it first seems. Using a pruned tree to obtain a rule instead of building it incrementally by adding conjunctions one at a time avoids the over-pruning problem of the basic separate-and-conquer rule learner. Using the separate-and-conquer methodology in conjunction with decision trees adds flexibility and speed. It is indeed wasteful to build a full decision tree just to obtain a single rule, but the process can be accelerated significantly without sacrificing the above advantages.

The key idea is to build a "partial" decision tree instead of a fully explored one. A partial decision tree is an ordinary decision tree that contains branches to undefined subtrees. To generate such a tree, we integrate the construction and pruning operations in order to find a "stable" subtree that can be simplified no further. Once this subtree has been found, tree-building ceases and a single rule is read off.

The tree-building algorithm is summarized in Figure 2: it splits a set of examples recursively into a partial tree. The first step chooses a test and divides the examples into subsets accordingly. Our implementation makes this choice in exactly the same way as C4.5. Then the subsets are expanded in order of their average entropy, starting with the smallest. (The reason for this is that subsequent subsets will most likely not end up being expanded, and the subset with low average entropy is more likely to result in a small subtree and therefore produce a more general rule.) This continues recursively until a subset is expanded into a leaf, and

Procedure Expand Subset

choose split of given set of examples into subsets
while there are subsets that have not been expanded **and**
 all the subsets expanded so far are leaves
 choose next subset to be expanded and expand it
if all the subsets expanded are leaves **and**
 estimated error for subtree \geq estimated error for node
 undo expansion into subsets and make node a leaf

Figure 2: Method that expands a given set of examples into a partial tree

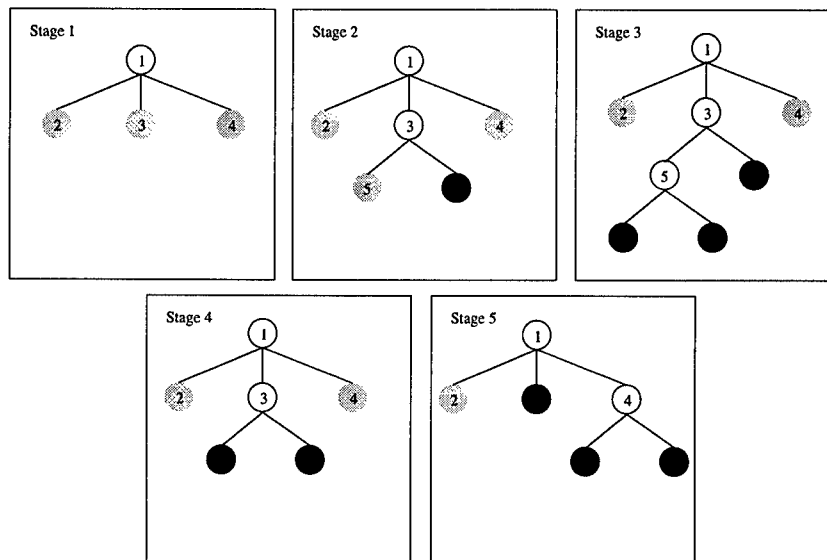


Figure 3: Example of how our algorithm builds a partial tree

then continues further by backtracking. But as soon as an internal node appears which has all its children expanded into leaves, pruning begins: the algorithm checks whether that node is better replaced by a single leaf. This is just the standard “subtree replacement” operation of decision-tree pruning, and our implementation makes the decision in exactly the same way as C4.5. (C4.5’s other pruning operation, “subtree raising,” plays no part in our algorithm.) If replacement is performed the algorithm backtracks in the standard way, exploring siblings of the newly-replaced node. However, if during backtracking a node is encountered all of whose children are not leaves—and this will happen as soon as a potential subtree replacement is *not* performed—then the remaining subsets are left unexplored and the corresponding subtrees are left undefined. Due to the recursive structure of the algorithm this event automatically terminates tree generation.

Figure 3 shows a step-by-step example. During stages 1–3, tree-building continues recursively in the normal way—except that at each point the lowest-entropy sibling is chosen for expansion: node 3 between stages 1 and 2. Gray nodes are as yet unexpanded; black ones are leaves. Between Stages 2 and 3, the black node will have lower entropy than its sibling, node 5; but cannot be expanded further since it is a leaf. Backtracking occurs and node 5 is chosen for expansion. Once stage 3 is reached, there is a node—node 5—which has all of its children expanded into leaves, and this triggers pruning. Subtree replacement for node 5 is considered, and accepted, leading to stage 4. Now node 3 is considered for subtree replacement, and this operation is again accepted. Backtracking continues, and node 4, having lower entropy than 2, is expanded—into two leaves. Now subtree replacement is considered for node 4: let us suppose that node 4 is not replaced. At this point, the process effectively terminates with the 3-leaf

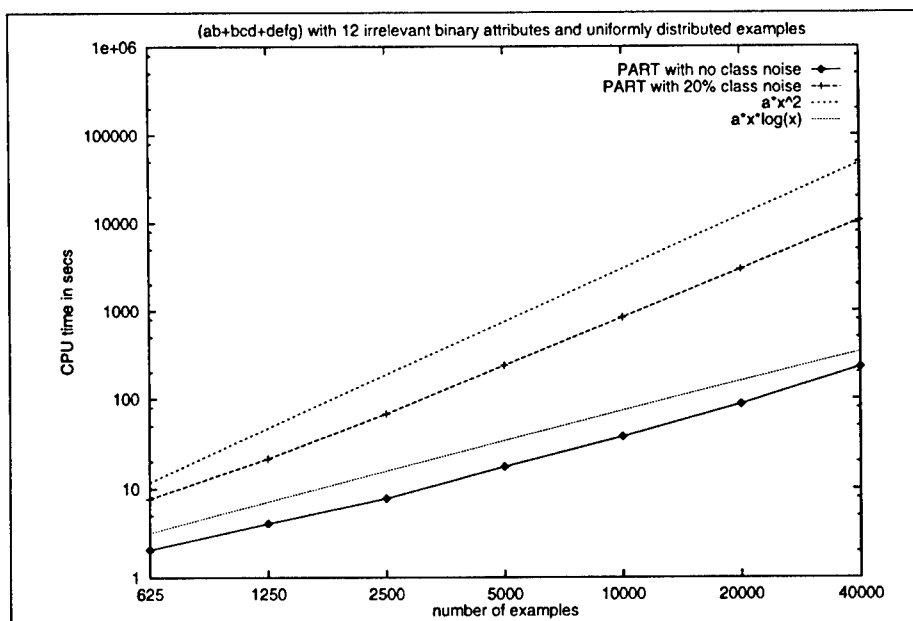


Figure 4: CPU times for PART on artificial dataset

partial tree of stage 5.

This procedure ensures that the over-pruning effect discussed in Section 2 cannot occur. A node can only be pruned if all its successors are leaves. This can only happen if all its subtrees have been explored and either found to be leaves, or are pruned back to leaves. Situations like that shown in Figure 1 are therefore handled correctly.

If a dataset is noise-free and contains enough instances to prevent the algorithm from doing any pruning, just one path of the full decision tree has to be explored. This achieves the greatest possible performance gain over the naive method that builds a full decision tree each time. The gain decreases as more pruning takes place. For datasets with numeric attributes, the asymptotic time complexity of the algorithm is the same as for building the full decision tree² because in this case the complexity is dominated by the time needed to sort the attribute values in the first place.

Once a partial tree has been built, a single rule is extracted from it. Each leaf corresponds to a possible rule, and we seek the "best" leaf of those subtrees (typically a small minority) that have been expanded into leaves. Our implementation aims at the most general rule by choosing the leaf that covers the greatest number of instances. (We have experimented with choosing

the most accurate rule, that is, the leaf with the lowest error rate, error being estimated according to C4.5's Bernoulli heuristic, but this does not improve the rule set's accuracy.)

Datasets often contain missing attribute values, and practical learning schemes must deal with them efficiently. When constructing a partial tree we treat missing values in exactly the same way as C4.5: if an instance cannot be assigned deterministically to a branch because of a missing attribute value, it is assigned to each of the branches with a weight proportional to the number of training instances going down that branch, normalized by the total number of training instances with known values at the node. During testing we apply the same procedure separately to each rule, thus associating a weight with the application of each rule to the test instance. That weight is deducted from the instance's total weight before it is passed to the next rule in the list. Once the weight has reduced to zero, the predicted class probabilities are combined into a final classification according to the weights.

The algorithm's runtime depends on the number of rules it generates. Because a decision tree can be built in time $O(an \log n)$ for a dataset with n examples and a attributes, the time taken to generate a rule set of size k is $O(kan \log n)$. Assuming (as the analyses of (Cohen, 1995) and (Fürnkranz, 1997) do) that the size of the final theory is constant, the overall time complexity is $O(an \log n)$, as compared to

²Assuming no subtree raising.

Table 1: Datasets used for the experiments

Dataset	Instances	Missing values (%)	Numeric attributes	Nominal attributes	Classes
anneal	898	0.0	6	32	5
audiology	226	2.0	0	69	24
australian	690	0.6	6	9	2
autos	205	1.1	15	10	6
balance-scale	625	0.0	4	0	3
breast-cancer	286	0.3	0	9	2
breast-w	699	0.3	9	0	2
german	1000	0.0	7	13	2
glass (G2)	163	0.0	9	0	2
glass	214	0.0	9	0	6
heart-c	303	0.2	6	7	2
heart-h	294	20.4	6	7	2
heart-statlog	270	0.0	13	0	2
hepatitis	155	5.6	6	13	2
horse-colic	368	23.8	7	15	2
hypothyroid	3772	5.5	7	22	4
ionosphere	351	0.0	34	0	2
iris	150	0.0	4	0	3
kr-vs-kp	3196	0.0	0	36	2
labor	57	3.9	8	8	2
lymphography	148	0.0	3	15	4
mushroom	8124	1.4	0	22	2
pima-indians	768	0.0	8	0	2
primary-tumor	339	3.9	0	17	21
segment	2310	0.0	19	0	7
sick	3772	5.5	7	22	2
sonar	208	0.0	60	0	2
soybean	683	9.8	0	25	19
splice	3190	0.0	0	61	3
vehicle	846	0.0	18	0	4
vote	435	5.6	0	16	2
vowel	990	0.0	10	3	11
waveform-noise	5000	0.0	40	0	3
zoo	101	0.0	1	15	7

$O(an \log^2 n)$ for RIPPER. In practice, the number of rules grows with the size of the training data because of the greedy rule learning strategy and pessimistic pruning. However, even in the worst case when the number of rules increases linearly with training examples, the overall complexity is bounded by $O(an^2 \log n)$. In our experiments we only ever observed subquadratic run times—even for the artificial dataset that Cohen (1995) used to show that C4.5's performance can be cubic in the number of examples. The results of timing our method, PART, on this dataset are depicted on a log-log scale in Figure 4, for no class noise and for 20 percent class noise. In the latter case C4.5 scales as the cube of the number of examples.

4 Experimental Results

In order to evaluate the performance of PART on a diverse set of practical learning problems, we performed experiments on thirty-four standard datasets from the UCI collection (Merz & Murphy, 1996).³ The datasets and their characteristics are listed in Table 1.

As well as the learning algorithm PART described above, we also ran C4.5,⁴ C5.0 and RIPPER on all the datasets. The results are listed in Table 2. They give the percentage of correct classifications, averaged over ten ten-fold cross-validation runs, and standard

³Following Holte (Holte, 1993), the G2 variant of the *glass* dataset has classes 1 and 3 combined and classes 4 to 7 deleted, and the *horse-colic* dataset has attributes 3, 25, 26, 27, 28 deleted with attribute 24 being used as the class. We also deleted all identifier attributes from the datasets.

⁴We used Revision 8 of C4.5.

Table 2: Experimental results: percentage of correct classifications, and standard deviation

Dataset	PART	C4.5		C5.0		RIPPER
anneal	98.4±0.3	98.6±0.2	†	98.7±0.3	○	98.3±0.1
audiology	78.7±1.1	76.3±1.2	●†	77.3±1.2	†	72.3±2.2 ●
australian	84.3±1.2	84.8±1.1	●	85.4±0.7		85.3±0.7
autos	74.5±1.4	76.5±2.9	†	79.1±2.1	○†	72.0±2.0 ●
balance-scale	82.3±1.2	78.0±0.7	●	79.0±1.0	●	81.0±1.1
breast-cancer	69.6±1.6	70.3±1.6		73.6±1.6	○	71.8±1.6 ○
breast-w	94.9±0.4	95.5±0.6	†	95.5±0.3	○	95.6±0.7
horse-colic	84.4±0.8	83.0±0.6	●	85.0±0.5		85.0±0.8
german	70.0±1.4	71.9±1.4	○	72.3±0.5	○	71.4±0.7 ○
glass (G2)	80.0±3.6	79.4±2.3	†	80.2±1.8	†	80.9±1.4
glass	70.0±1.6	67.3±2.4		68.4±2.8	†	66.7±2.1 ●
heart-c	78.5±1.7	79.7±1.5		79.1±0.9		78.5±1.9
heart-h	80.5±1.5	79.7±1.7		80.7±1.1		78.7±1.3 ●
heart-statlog	78.9±1.3	81.2±1.3	○	81.9±1.4	○	79.0±1.4
hepatitis	80.2±1.9	79.7±1.0	†	81.1±0.7		77.2±2.0 ●
hypothyroid	99.5±0.1	99.5±0.1	†	99.5±0.0	●†	99.4±0.1 ●
ionosphere	90.6±1.3	89.9±1.5	†	89.3±1.4	●†	89.2±0.8 ●
iris	93.7±1.6	95.1±1.0	○†	94.4±0.7	†	94.4±1.7
kr-vs-kp	99.3±0.1	99.4±0.1	○†	99.3±0.1		99.1±0.1 ●
labor	77.3±3.9	81.4±2.6	○†	77.1±3.7	†	83.5±3.9 ○
lymphography	76.5±2.7	78.0±2.2		76.8±2.7	†	76.1±2.4
mushroom	100.0±0.0	100.0±0.0	●†	99.9±0.0	●†	100.0±0.0
pima-indians	74.0±0.5	74.2±1.2	†	75.5±0.9	○†	75.2±1.1 ○
primary-tumor	41.7±1.3	40.1±1.7	●	28.7±2.5	●	38.5±0.8 ●
segment	96.6±0.4	96.1±0.3	●†	96.3±0.4	†	95.2±0.5 ●
sick	98.6±0.1	98.4±0.2	●	98.4±0.1	●	98.3±0.2 ●
sonar	76.5±2.3	74.4±2.9	†	75.3±2.2	†	75.7±1.9
soybean	91.4±0.5	91.9±0.7		92.2±0.6	†	92.0±0.4
splice	92.5±0.4	93.4±0.3	○	94.3±0.3	○	93.4±0.2 ○
vehicle	72.4±0.8	72.9±0.9		72.4±0.8	†	69.0±0.6 ●
vote	95.9±0.6	95.9±0.6	†	96.0±0.6	†	95.6±0.3
vowel	78.1±1.1	77.9±1.3	†	79.9±1.2	○†	69.6±1.9 ●
waveform-noise	78.0±0.5	76.3±0.4	●	79.4±0.5	○	79.1±0.6 ○
zoo	92.2±1.2	90.9±1.2	●†	91.5±1.2	†	87.8±2.4 ●

deviations of the ten are also shown. The same folds were used for each scheme.⁵ Results for C4.5, C5.0 and RIPPER are marked with ○ if they show significant improvement over the corresponding results for PART, and with ● if they show significant degradation. (The † marks are discussed below.) Throughout, we speak of results being “significantly different” if the difference is statistically significant at the 1% level according to a paired two-sided *t*-test, each pair of data points consisting of the estimates obtained in one ten-fold cross-validation run for the two learning schemes being compared. Table 3 shows how the different methods compare with each other. Each entry

indicates the number of datasets for which the method associated with its column is significantly more accurate than the method associated with its row.

We observe from Table 3 that PART outperforms C4.5 on nine datasets, whereas C4.5 outperforms PART on six. The chance probability of this distribution is 0.3 according to a sign test: thus there is only very weak evidence that PART outperforms C4.5 on a collection of datasets similar to the one we used. According to Table 3, PART is significantly less accurate than C5.0 on ten datasets and significantly more accurate on six. The corresponding probability for this distribution is 0.23, providing only weak evidence that C5.0 performs better than PART. For RIPPER the situation is different: PART outperforms it on fourteen datasets and performs worse on six. The probability for this distribution is 0.06, a value that provides fairly strong

⁵The results of PART and C5.0 on the hypothyroid data, and of PART and C4.5 on the mushroom data, are not in fact the same—they differ in the second decimal place.

evidence that PART outperforms RIPPER on a collection of datasets of this type.

Table 3: Results of paired t -tests ($p=0.01$): number indicates how often method in column significantly outperforms method in row

	PART	C4.5	C5.0	RIPPER
PART	—	6	10	6
C4.5	9	—	9	4
C5.0	6	5	—	4
RIPPER	14	10	12	—

As well as accuracy, the size of a rule set is important because it has a strong influence on comprehensibility. The † marks in Table 2 give information about the relative size of the rule sets produced: they mark learning schemes and datasets for which—on average—PART generates fewer rules (this never occurs for RIPPER). Compared to C4.5 and C5.0, the average number of rules generated by PART is smaller for eighteen datasets and larger for sixteen.

5 Conclusions

This paper has presented a simple, yet surprisingly effective, method for learning decision lists based on the repeated generation of partial decision trees in a separate-and-conquer manner. The main advantage of PART over the other schemes discussed is not performance but simplicity: by combining two paradigms of rule learning it produces good rule sets without any need for global optimization. Despite this simplicity, the method produces rule sets that compare favorably with those generated by C4.5 and C5.0, and are more accurate (though larger) than those produced by RIPPER.

An interesting question for future research is whether the size of the rule sets obtained by our method can be decreased by employing a stopping criterion based on the minimum description length principle, as is done in RIPPER, or by using reduced error pruning instead of pessimistic pruning.

Acknowledgements

We would like to thank the anonymous reviewers for their comments, which significantly improved the exposition. We would also like to thank all the people who have donated datasets to the UCI repository.

References

- Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning* (pp. 115–123). Morgan Kaufmann.
- Fürnkranz, J. (1996). Separate-and-conquer rule learning. Technical Report TR-96-25, Austrian Research Institute for Artificial Intelligence, Vienna. [[ftp://ftp.ai.univie.ac.at/papers/oefai-tr-96-25.ps.Z](http://ftp.ai.univie.ac.at/papers/oefai-tr-96-25.ps.Z)].
- Fürnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning*, 27(2), 139–171.
- Fürnkranz, J. & Widmer, G. (1994). Incremental reduced error pruning. In *Proceedings of the 11th International Conference on Machine Learning* (pp. 70–77). Morgan Kaufmann.
- Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11, 63–91.
- Merz, C. J. & Murphy, P. M. (1996). *UCI Repository of Machine Learning Data-Bases*. Irvine, CA: University of California, Department of Information and Computer Science. [<http://www.ics.uci.edu/~mlearn/MLRepository.html>].
- Michalski, R. S. (1969). On the quasi-minimal solution of the covering problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, Vol. A3 (*Switching Circuits*) (pp. 125–128). Bled, Yugoslavia.
- Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 71–99.
- Quinlan, J. R. (1987a). Generating production rules from decision trees. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (pp. 304–307). Morgan Kaufmann.
- Quinlan, J. R. (1987b). Simplifying decision trees. *International Journal of Man-Machine Studies*, 27, 221–234.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- Rissanen, J. (1978). Modelling by shortest data description. *Automatica*, 14, 465–471.
- Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, 2, 229–246.

Using a Permutation Test for Attribute Selection in Decision Trees

Eibe Frank

Department of Computer Science
University of Waikato
Hamilton, New Zealand
eibe@cs.waikato.ac.nz

Ian H. Witten

Department of Computer Science
University of Waikato
Hamilton, New Zealand
ihw@cs.waikato.ac.nz

Abstract

Most techniques for attribute selection in decision trees are biased towards attributes with many values, and several *ad hoc* solutions to this problem have appeared in the machine learning literature. Statistical tests for the existence of an association with a prespecified significance level provide a well-founded basis for addressing the problem. However, many statistical tests are computed from a chi-squared distribution, which is only a valid approximation to the actual distribution in the large-sample case—and this patently does not hold near the leaves of a decision tree. An exception is the class of permutation tests. We describe how permutation tests can be applied to this problem. We choose one such test for further exploration, and give a novel two-stage method for applying it to select attributes in a decision tree. Results on practical datasets compare favorably with other methods that also adopt a pre-pruning strategy.

1 Introduction

Statistical tests provide a set of theoretically well-founded tools for testing hypotheses about relationships in a set of data. One pertinent hypothesis, when selecting attributes for a decision tree, is whether there is a significant association between an attribute's values and the classes. With r attribute values and c classes, this equates to testing for independence in the corresponding $r \times c$ contingency table (White & Liu, 1994), and statistical tests designed for this purpose can be applied directly. Unlike most commonly-used

attribute selection criteria, such tests are not biased towards attributes with many values, which is important because it prevents the decision tree induction algorithm from selecting splits that overfit the training data by being too fine-grained.

Statistical tests are based on probabilities derived from the distribution of a test statistic. Two popular test statistics for assessing independence in a contingency table have been proposed for attribute selection: the chi-squared statistic χ^2 and the log likelihood ratio G_2 (White & Liu, 1994). For large samples, both are distributed according to the chi-squared distribution. But this is not the case for small samples (Agresti, 1990)—and small samples inevitably occur close to the leaves in a decision tree. Thus it is inadvisable to use probabilities derived using the chi-squared distribution for decision tree induction.

Fortunately, there is an alternative that does apply in small frequency domains. In statistical tests known as “permutation tests” (Good, 1994), the distribution of the statistic of interest is calculated directly instead of relying on the chi-squared approximation—in other words they are “non-parametric” rather than “parametric.” Such tests do not suffer from the small expected frequency problem because they do not use the chi-squared approximation.

This paper describes the application of permutation tests to attribute selection in a decision tree. We examine one such test—the Freeman and Halton test—in detail by performing experiments on artificial and practical datasets: the results show that this method is indeed preferable to a test that assumes the chi-squared distribution. The statistic of the Freeman and Halton test is the exact probability p_f of a contingency table f given its marginal totals (Good, 1994). Recently, Martin (1997) investigated the use of this statistic, p_f , directly for attribute selection. We show

that results can be improved by using it in conjunction with the Freeman and Halton test.

Section 2 introduces the idea of permutation tests and how they can be used to test significance in a contingency table. In Section 2.2 we describe the Freeman and Halton test. The test is expensive, but simple computational economies are described in Section 2.3. Section 2.4 describes a novel two-stage method, based on these ideas, for selecting attributes in a decision tree. Section 3 presents experimental results on artificial and standard datasets. We verify that the Freeman and Halton test does not prefer attributes with many values, whereas the test statistic p_f by itself is biased. We also verify that the parametric version of the chi-squared test is biased in small-frequency domains. Finally, we demonstrate that good results are obtained when the new method is applied to decision-tree building. Section 4 reviews existing work on using statistical tests for contingency tables in machine learning, while Section 5 contains some concluding remarks.

2 A Permutation Test and its Application to Attribute Selection

The procedure for permutation tests is simple (Good, 1994). First, a test statistic is chosen that measures the strength of the effect being investigated, and is computed over the data. The null hypothesis is that the observed strength of the effect is *not* significant. Next, the labels of the original data are permuted and the same statistic is calculated for the relabeled data; this is repeated for all possible permutations of labels. The idea is to ascertain the likelihood of an effect of the same or greater strength being observed fortuitously on randomly labeled data with identical marginal properties. Third, the test statistic's value for the original data is compared with the values obtained over all permutations, by calculating the percentage of the latter that are at least as extreme, or more extreme, than the former. This percentage constitutes the significance level at which the null hypothesis can be rejected, in other words, the level at which the observed strength of the effect can be considered significant.

2.1 Permutation Tests for Contingency Tables

Contingency tables summarize the observed relationship between two categorical response variables. Several different statistics can be used to measure the

strength of the dependency between two variables (Good, 1994), the two most common being the chi-squared statistic χ^2 and the log likelihood ratio G_2 . The standard tests using these statistics are based on the fact that the sampling distribution of both statistics is well-approximated by the chi-squared distribution. They calculate the significance level directly from that distribution.

Unfortunately, as noted in the introduction, the chi-squared distribution assumption is only valid for either statistic when the sample size is large enough. The chi-squared distribution approximates the true sampling distribution poorly if the sample size is small (or the samples are distributed unevenly in the contingency table). In a decision tree the sample size becomes smaller and smaller and the distribution of the samples more and more skewed the closer one gets to the leaves of the tree. Thus one cannot justify using a test based on the chi-squared approximation for significance testing throughout a decision tree (although one might at the upper levels where samples are large). Permutation tests offer a theoretically sound alternative that is admissible for any sample size.

The standard permutation test for $r \times c$ contingency tables, which we have also chosen to employ for this paper, is based on the statistic p_f , the exact probability of a contingency table given its marginal totals. It is known as the "Freeman and Halton" test and it is a generalization of Fisher's exact test for 2×2 tables (Good, 1994). However, we emphasize that other test statistics could equally well be used, thereby obtaining exact, non-parametric, versions of conventional parametric tests that are valid in small-frequency domains (Good, 1994).¹

2.2 Testing the Significance of an Attribute

For attribute selection, we seek to test whether there is a significant association between an attribute's values and the class values. With r attribute values and c classes, this is the same as testing for independence in the corresponding $r \times c$ contingency table (White & Liu, 1994).

If the $r \times c$ contingency table f contains the frequencies f_{ij} with column marginals $f_{.j}$ and row marginals $f_{i.}$, the probability p_f of this table is given by

¹We have also used a permutation test based on χ^2 , instead of on p_f , in all the experiments described in Section 3, and obtained almost identical results.

$$p_f = \frac{\prod_{i=1}^r f_{i.}! \prod_{j=1}^c f_{.j}!}{f_{..}! \prod_{i=1}^r \prod_{j=1}^c f_{ij}!}.$$

Permuting the instances' class labels does not affect the row and column totals, and therefore the set of all permutations of the class labels corresponds to the set of all contingency tables with the same row and column totals. If p is the proportion of tables for which p_f is less than or equal to the probability p_o of the original table, then

$$p = \sum I(p_f \leq p_o) p_f,$$

where $I(\cdot)$ denotes the indicator function, constitutes the p -value of the Freeman and Halton test. The function computing p is known as a multiple hypergeometric distribution (Agresti, 1990). The resulting value of p is simply compared with a prespecified desired significance level.

2.3 Approximating the Exact Test

Exact computation of the p -value of a permutation test is only possible for sparsely populated tables, and is computationally infeasible for most tables resulting from practical machine learning datasets. Fortunately, p can be approximated to arbitrary precision by Monte Carlo sampling as follows (Good, 1994).

For each of n trials the class labels are randomly permuted, the test statistic is computed, and its value is compared to the value for the original (unpermuted) data. The percentage of trials for which the artificially generated value is less than or equal to the original value constitutes an estimate \hat{p} of the exact significance level p . This estimate is a binomial random variable with standard error $se(\hat{p}) = \sqrt{\hat{p}(1 - \hat{p})/n}$, and so its $100(1 - \alpha)\%$ confidence interval is $\hat{p} \pm t_{n-1}(\alpha/2)se(\hat{p})$, where $t_{n-1}(\alpha/2)$ is obtained from Student's t -distribution.

This information is used to decide when to stop performing trials. Let p_{fixed} be the prespecified desired minimum significance level that an attribute must achieve unless it is to be considered independent of the class—the level at which the null hypothesis of “no significant dependence” is to be rejected. Then, with probability $(1 - \alpha)$,

$$p > p_{\text{fixed}} \quad \text{if} \quad p_{\text{fixed}} < \hat{p} - t_{n-1}(\alpha)se(\hat{p}),$$

and

$$p < p_{\text{fixed}} \quad \text{if} \quad p_{\text{fixed}} > \hat{p} + t_{n-1}(\alpha)se(\hat{p}).$$

If the first inequality holds we judge the attribute to be significant; if the second holds we do not.² As n increases, the likelihood that one of the two inequalities will be true increases, but if p is very close to p_{fixed} , neither inequality will become true in a reasonable amount of time. Therefore the procedure is terminated when the number of trials reaches a pre-specified maximum,³ and any attribute that survives this number of trials is considered significant. The introduction of this cut-off point slightly increases the probability that an attribute is incorrectly judged to be significant.

2.4 Procedure for Attribute Selection

At each node of a decision tree we must decide which attribute to split on. This is done in two steps. First, attributes are rejected if they show no significant association to the class according to a pre-specified significance level. To judge “significance” we employ the Freeman and Halton test, approximated by Monte Carlo sampling as described above. Second, from the attributes that remain, the one with the lowest value of p_f is chosen.⁴ The selected attribute is then used to split the set of instances, and the algorithm recurses.

The division into two steps is a crucial part of the procedure. It distinguishes clearly between the different concepts of *significance* and *strength*. For example, it is well known that the association between two distributions may be very significant even if that association is weak—if the quantity of data is large enough (Press, Teukolsky, Vetterling & Flannery, 1988, p. 628). First, we test the significance of an association using a permutation test (specifically, the Freeman and Halton test); then we consider its strength (as measured by the exact probability p_f).

If no significant attributes are found in the first step, the splitting process stops and the subtree is not expanded any further. This gives an elegant, uniform, technique for pre-pruning.

3 Experimental Results

We begin with two controlled experiments that are designed to verify the relative performance of (a) the use

²Here, α is used instead of $\alpha/2$ because the comparisons are one-sided. In our experiments we set α to 0.005.

³We use at least 100 and at most 1000 trials in our experiments.

⁴Other attribute selection criteria could be employed at this stage; p_f was chosen to allow a direct comparison with the method proposed by Martin (1997).

Table 1: Average probabilities for random data (600 instances; uniformly distributed attribute values)

Attribute Values	Class Values	(a) \hat{p}	(b) p_f	(c) p_χ
2	2	0.525	0.045	0.488
2	5	0.511	1.63e-05	0.509
2	10	0.506	1.80e-10	0.505
5	2	0.497	1.55e-05	0.496
5	5	0.500	9.25e-18	0.497
5	10	0.491	6.62e-35	0.487
10	2	0.498	1.77e-10	0.495
10	5	0.520	7.84e-35	0.515
10	10	0.512	4.89e-68	0.503

Table 2: Average probabilities for random data (20 instances; non-uniformly distributed attribute values)

Attribute Values	Class Values	(a) \hat{p}	(b) p_f	(c) p_χ
2	2	0.745	0.285	0.515
2	5	0.674	0.024	0.466
2	10	0.741	0.004	0.446
5	2	0.549	0.027	0.444
5	5	0.561	1.02e-4	0.448
5	10	0.632	1.80e-6	0.418
10	2	0.548	0.004	0.430
10	5	0.581	1.72e-6	0.425
10	10	0.639	1.42e-8	0.382

of the exact-probability p_f statistic in the Freeman and Halton test, (b) the use of p_f by itself with no significance test (Martin, 1997), and (c) the use of the parametric version of the chi-squared test, that is, the probability of χ^2 calculated from the chi-squared distribution (White & Liu, 1994). The first experiment exhibits an artificial dataset for which method (b) performs poorly because it is biased towards many-valued attributes, whereas (a) performs well (and so does (c)). The second exhibits another dataset for which method (c) is biased towards many-valued attributes and performs poorly (and (b) performs even worse), whereas (a) continues to perform well.

The third subsection presents results for building decision trees on practical datasets using the new method.

3.1 Using the Exact Probability p_f is Biased

In order to show that the exact probability p_f is biased towards attributes with many values, we adopt the experimental setup of White and Liu (1994). This involves an artificial dataset that exhibits no actual association between class and attribute values. For each class, an equal number (300) of instances with random, uniformly distributed attribute values is generated. The estimated p -value of the Freeman and Halton test \hat{p} , the exact probability p_f , and the p -value of

the parametric chi-squared test p_χ are calculated for this artificial, non-informative, attribute.⁵ This procedure is repeated 1000 times with different random seeds used to generate the instances.

Table 1 shows the average values obtained. It can be seen in column (b) that p_f systematically decreases with increasing number of classes and attribute values. Even more importantly, it is always close to zero. If used for pre-pruning at the 0.01 level (as proposed by Martin, 1997), it would fail to stop splitting in every situation except that represented by the first row. On the other hand, neither \hat{p} nor p_χ varies systematically with the number of attribute and class values. For these reasons it is inadvisable to use p_f for attribute selection without preceding it with a significance test.

3.2 Parametric Chi-Squared Test is Biased

A similar experimental procedure was used to show that the parametric chi-squared test is biased in small frequency domains with unevenly distributed samples. Instead of generating the attribute values uniformly, they are skewed so that more samples lie close to the zero point. This is done using the distribution $[kx^2]$, where k is the number of attribute values and x is

⁵Our experiments use $N = 1000$ Monte Carlo trials to estimate \hat{p} .

distributed uniformly between 0 and 1. The number of instances is reduced to twenty.

Table 2 shows the average values obtained using this procedure. It can be seen that p_χ decreases systematically as the number of attribute values increases, whereas this is not the case for \hat{p} . The test based on p_χ is too liberal in this situation. There also exist situations in which it is too conservative (Good, 1994). If used for pruning in a decision tree, a test that is too liberal does not prune enough, and a test that is too conservative prunes too much.

3.3 Comparison on Practical Datasets

Results are now presented for building decision trees for thirty-one UCI datasets (Merz & Murphy, 1996) using the method described above. We eliminated missing values from the datasets by deleting all attributes with more than 10% missing values, and subsequently removing all instances with missing values. The resulting datasets are summarized in Table 3. All numeric attributes were discretized into four intervals of equal width.⁶

We compare pre-pruned trees built using (a) p_f with prior significance testing using the Freeman and Halton test \hat{p} , (b) the exact probability p_f , (c) p_f with prior significance testing using the parametric chi-squared test p_χ , and (d) post-pruned trees built using C4.5's pessimistic pruning with default parameter settings (Quinlan, 1993). We also include results for pruned and unpruned trees as built by C4.5. Note that for (a) and (c) we are now applying the two-step attribute selection procedure developed in Section 2.4, first discarding insignificant attributes and then selecting the best among the remainder. Results are reported for three significance levels: 0.01, 0.05 and 0.10. All results were generated using ten-fold cross-validation repeated ten times with different randomizations of the dataset. The same folds were used for each scheme.⁷

Table 4 shows how method (a) compares with the others. Each row contains the number of datasets for which it builds significantly more (+) or less (−) accurate trees, and significantly smaller (+) or larger (−) trees than the method associated with this row. We speak of results being “significantly different” if the

Table 3: Datasets used for the experiments

Dataset	Size	Attributes (numeric/total)	Classes
anneal	898	6/38	5
audiology	216	0/67	24
australian	653	6/15	2
autos	193	14/24	6
balance-scale	625	4/ 4	3
breast-cancer	277	0/ 9	2
breast-w	683	9/ 9	2
german	1000	7/20	2
glass (G2)	163	9/ 9	2
glass	214	9/ 9	6
heart-c	296	6/13	2
heart-h	261	5/10	2
heart-statlog	270	13/13	2
hepatitis	137	3/16	2
hypothyroid	3404	2/24	4
ionosphere	351	34/34	2
iris	150	4/ 4	3
kr-vs-kp	3196	0/36	2
lymphography	148	3/18	4
mushroom	8124	0/21	2
pima-indians	768	8/ 8	2
primary-tumor	336	0/15	21
segment	2310	19/19	7
sick	3404	2/24	2
sonar	208	60/60	2
soybean	630	0/16	15
splice	3190	0/61	3
vehicle	846	18/18	4
vote	312	0/15	2
vowel	990	10/13	11
zoo	101	1/16	7

difference is statistically significant at the 1% level according to a paired two-sided t -test, each pair of data points consisting of the estimates obtained in one ten-fold cross-validation run for the two learning schemes being compared. Results are shown for three different significance levels: note that this refers to the level at which attributes are rejected prior to the selection process.

Observe first that pre-pruning using \hat{p} outperforms pre-pruning using p_f (the three rows marked (b)), confirming our findings from Section 3.1. For all three significance levels \hat{p} dominates p_f in both accuracy and size of the trees produced. These results show that if the splitting attribute is selected based on the value of p_f , it is better to use a significance test first.

One might think that p_f performs poorly with respect to \hat{p} because the former does not prune sufficiently—it is inferior in terms of both accuracy and tree size. Consequently, we also ran pre-pruning using p_f at the 0.005 and 0.001 levels, and found that the performance

⁶If the class information were used when discretizing the attributes, the assumptions of the statistical tests would be invalidated.

⁷Appendix A lists the average accuracy and standard deviation for a representative subset of the methods.

Table 4: Number of times \hat{p} performs significantly better (+) or worse (−) than (b) p_f , (c) p_χ , (d) post-pruned trees, and pruned and unpruned C4.5 trees with respect to accuracy and tree size

		Accuracy		Tree Size	
	\hat{p}	+	−	+	−
$p_{\text{fixed}} = 0.01$	(b) p_f	8	5	17	6
	(c) p_χ	9	3	8	11
	(d) post-pruned	4	14	20	7
	C4.5 pruned	3	17	20	7
	C4.5 unpruned	11	11	31	0
$p_{\text{fixed}} = 0.05$	(b) p_f	8	2	22	3
	(c) p_χ	6	6	24	2
	(d) post-pruned	4	9	8	17
	C4.5 pruned	2	16	11	15
	C4.5 unpruned	8	9	29	2
$p_{\text{fixed}} = 0.1$	(b) p_f	9	2	24	1
	(c) p_χ	5	5	24	0
	(d) post-pruned	4	12	5	22
	C4.5 pruned	3	16	3	24
	C4.5 unpruned	8	8	29	2

Table 5: Number of times \hat{p} with gain ratio (Method a') performs significantly better (+) or worse (−) than \hat{p} with p_f (Method a), and pruned and unpruned C4.5 trees

		Accuracy		Tree Size	
	\hat{p} with gain ratio	+	−	+	−
$p_{\text{fixed}} = 0.01$	\hat{p} with p_f	8	3	10	10
	C4.5 pruned	3	14	21	6
	C4.5 unpruned	13	7	30	1
$p_{\text{fixed}} = 0.05$	\hat{p} with p_f	10	4	11	14
	C4.5 pruned	0	10	10	14
	C4.5 unpruned	12	7	30	1
$p_{\text{fixed}} = 0.1$	\hat{p} with p_f	10	5	11	12
	C4.5 pruned	1	15	6	22
	C4.5 unpruned	13	8	30	0

difference between p_f and \hat{p} can not be eliminated by adjusting the significance level.

Next, observe from the three rows marked (c) that for the 0.01 significance level, pre-pruning using \hat{p} beats pre-pruning using p_χ with respect to the accuracy of the resulting trees. For this significance level the two methods produce trees of similar size. However, for both the 0.05 and the 0.1 levels \hat{p} produces trees that are significantly smaller than those produced by p_χ . For these two significance levels the two methods perform comparably as far as accuracy is concerned. These facts indicate that for both the 0.05 and the 0.1 levels p_χ is a more liberal test than \hat{p} if applied to attribute selection and pre-pruning; p_χ stops later than \hat{p} —as for the artificial dataset used in Section

3.2. However, it is sometimes more conservative—in particular for the 0.01 level. The two tests really do behave differently: they cannot be forced to behave in the same way by adjusting their significance levels. However, the results show that trees produced by \hat{p} are preferable to those produced by p_χ .

Table 4 also shows that post-pruning consistently beats pre-pruning using \hat{p} , so far as accuracy is concerned (rows marked (d)). Our findings show that all the investigated pre-pruning methods perform significantly worse than pessimistic post-pruning.⁸ For both the 0.01 and the 0.05 levels, there are five datasets

⁸This contradicts a previous result (Martin, 1997) that trees pre-pruned using p_f are as accurate as, and smaller than, trees post-pruned using pessimistic pruning.

on which *all* pre-pruning methods consistently perform significantly worse than post-pruning: hypothyroid, kr-vs-kp, sick, splice, and vowel. On kr-vs-kp and vowel the pre-pruning methods stop too early, on the other three they stop too late. This means that the problem cannot be solved by adjusting the significance level of the pre-pruning methods.

For reference Table 4 also includes results for pruned and unpruned decision trees built by C4.5. C4.5's method for building pruned trees differs from post-pruning method (d) only in that it employs the gain ratio⁹ instead of p_f for attribute selection.

Surprisingly, Table 4 shows that \hat{p} does not perform better than C4.5's unpruned trees as far as accuracy is concerned, although \hat{p} performs better than unpruned trees built using p_f (results not shown). This indicates that the gain ratio produces more accurate trees than p_f . We therefore replaced attribute selection using p_f in the second step of pre-pruning method (a) by selection based on the gain ratio. As Table 5 shows, the new method (a')—selection based on the gain ratio with prior significance testing using the Freeman and Halton test \hat{p} —indeed performs better than method (a), and it also outperforms C4.5's unpruned trees. However, as Table 5 also shows, post-pruning—in this case represented by C4.5's pruned trees—still consistently beats pre-pruning using \hat{p} .

4 Related Work

Several researchers have applied parametric statistical tests to attribute selection in decision trees (White & Liu, 1994; Kononenko, 1995) and proposed remedies for their shortcomings (Martin, 1997). These are reviewed in the next section. Following that we discuss work on permutation tests for machine learning, none of which has been concerned with attribute selection in decision trees.

4.1 Use of Statistical Tests for Attribute Selection

White and Liu (1994) compare several entropy-based selection criteria to parametric tests that rely on the chi-squared distribution. More specifically, they compared the entropy-based measures to parametric tests based on both the chi-squared and log likelihood ratio statistics. They conclude that each of the entropy

⁹More precisely, it selects the attribute with maximum gain ratio among the attributes with more than average information gain.

measures favors attributes with larger numbers of values, whereas the statistical tests do not suffer from this problem. However, they also mention the problem of small expected frequencies with parametric tests and suggest the use of Fisher's exact test as a remedy. The extension of Fisher's exact test to $r \times c$ tables is the Freeman and Halton test that we have used above.

Kononenko (1995) repeated and extended these experiments and investigated several other attribute selection criteria as well. He shows that the parametric test based on the log likelihood ratio is biased towards attributes with many values if the number of classes and attribute values relative to the number of instances exceed the corresponding figures considered by White and Liu (1994). This is not surprising: it can be traced to the problem of small expected frequencies. For the log likelihood ratio the effect is more pronounced than for the chi-squared statistic (Agresti, 1990).

Kononenko also observes another problem with statistical tests. The restricted floating-point precision of most computer arithmetic makes it difficult to use them to discriminate between different *informative* attributes. The reason for this is that the association to the class is necessarily highly significant for all informative attributes.¹⁰ However, there is an obvious solution, which we pursue in this paper: once it has been established that an attribute is significant, it can be compared to other significant attributes using an attribute selection criterion that measures the strength of the association.

Recently, Martin (1997) used the exact probability of a contingency table given its marginal totals p_f for attribute selection and pre-pruning. Our method differs from his only in that we employ a significance test, based on p_f but not identical to it, to determine the significance of an attribute before selecting the best of the *significant* attributes according to p_f . As Section 3 of this paper establishes, direct use of p_f for attribute selection produces biased results.

4.2 Use of Permutation Tests in Machine Learning

Apparently the first to use a permutation test for machine learning, Gaines (1989) employs an approximation to Fisher's exact test to judge the quality of rules found by the INDUCT rule learner.¹¹ Instead of the

¹⁰The probability that the null hypothesis of no association between attribute and class values is incorrectly rejected is very close to zero.

¹¹He uses the one-tailed version of Fisher's exact test.

Figure 1: Two 2×2 -tables which both optimize the test statistic

3	0
0	3

0	3
3	0

hypergeometric distribution he uses the binomial distribution, which is a good approximation if the sample size is small relative to the population size (smaller than 10 percent).

Jensen (1992) gives an excellent introduction to permutation tests.¹² He discusses several alternatives, points out their weaknesses, and deploys the methodology in a prototypical rule learner. However, he does not mention the prime advantage of permutation tests, which makes them especially interesting in the context of decision trees: their applicability to small-frequency domains.

5 Conclusions

We have applied an approximate permutation test based on the multiple hypergeometric distribution to attribute selection and pre-pruning in decision trees, and explained why it is preferable to tests based on the chi-squared distribution. We have shown that using the exact probability of a contingency table given its marginal totals without a prior significance test is biased towards attributes with many values and performs worse in comparison. Although we were able to improve on existing methods for pre-pruning, we could not achieve the same accuracy as post-pruning.

Apart of the standard explanation that pre-pruning misses hidden attribute interactions, there are two other possible reasons for this result. The first is that we did not adjust for multiple comparisons when testing the significance of an attribute. Recently, Jensen and Schmill (1997) showed how to reduce the size of a post-pruned tree significantly by taking multiple hypotheses into account using a technique known as the "Bonferroni correction." The second reason is that tests for $r \times c$ contingency tables are inherently multi-sided. Consider the table shown at the left of Figure 1, which corresponds to a perfect classification of two classes using an attribute with two values. There is another permutation of class labels, shown at the right, that also results in a contingency table with the same optimum value of the test statistic. The significance

level achieved by the original table is only half as great as it would be if there were only one table that optimized the test statistic. In the case of two attributes and two classes, the one-sided version of Fisher's exact test avoids this problem. Generalizing this to the $r \times c$ case appears to be an open problem.

References

- Agresti, A. (1990). *Categorical Data Analysis*. New York: John Wiley & Sons.
- Gaines, B. (1989). An ounce of knowledge is worth a ton of data. In *Proceedings of the 6th International Workshop on Machine Learning* (pp. 156–159). Morgan Kaufmann.
- Good, P. (1994). *Permutation Tests*. New York: Springer-Verlag.
- Jensen, D. (1992). *Induction with Randomization Testing*. PhD thesis, Washington University, St. Louis, Missouri. [<http://eksl-www.cs.umass.edu/~jensen/papers/dissertation.ps>].
- Jensen, D. & Schmill, M. (1997). Adjusting for multiple comparisons in decision tree pruning. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. AAAI Press. [<http://eksl-www.cs.umass.edu/~jensen/papers/kdd97.ps>].
- Kononenko, I. (1995). On biases in estimating multi-valued attributes. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1034–1040). Morgan Kaufmann.
- Martin, J. K. (1997). An exact probability metric for decision tree splitting and stopping. *Machine Learning*, 28(2,3), 257–291.
- Merz, C. J. & Murphy, P. M. (1996). *UCI Repository of Machine Learning Data-Bases*. Irvine, CA: University of California, Department of Information and Computer Science. [<http://www.ics.uci.edu/~mlearn/MLRepository.html>].
- Press, W. H., Teukolsky, S., Vetterling, W. & Flannery, B. (1988). *Numerical Recipes in C* (2nd Ed.). Cambridge University Press.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- White, A. P. & Liu, W. Z. (1994). Bias in information-based measures in decision tree induction. *Machine Learning*, 15(3), 321–329.

¹²He uses the term "randomization test" instead of permutation test.

A Accuracy for Practical Datasets

Table 6: Experimental results: percentage of correct classifications, and standard deviation using \hat{p} , p_f , p_x , post-pruned trees, \hat{p} with gain ratio, C4.5's pruned trees, and C4.5's unpruned trees. Because of space constraints, we could only include results for one of the three p_{fixed} vales used in Table 4: we chose $p_{\text{fixed}} = 0.05$. In the last six columns, figures are marked with • if they are significantly worse than the corresponding results for \hat{p} , and with ◦ if they are significantly better.

	\hat{p}	p_f	p_x	post-pruned	\hat{p} with gain ratio	C4.5 pruned	C4.5 unpruned
anneal	98.6±0.1	98.5±0.0 •	99.0±0.1 ◦	98.4±0.1 •	98.3±0.3	98.0±0.3 •	98.3±0.3
audiology	71.6±1.9	70.3±1.9 •	71.5±1.7	71.9±1.3	73.8±1.2 ◦	74.8±1.0 ◦	74.8±1.3 ◦
australian	85.7±0.5	86.7±0.5 ◦	85.0±0.5 •	86.4±0.0 ◦	84.8±0.5 •	85.2±0.4	83.8±1.0 •
autos	67.3±2.2	67.2±2.4	72.7±2.4 ◦	70.5±2.4 ◦	73.3±2.3 ◦	73.0±2.0 ◦	72.9±2.3 ◦
balance-scale	66.1±0.9	70.5±1.2 ◦	65.9±1.2	67.3±1.0	67.2±1.2 ◦	67.9±1.0 ◦	74.1±1.0 ◦
breast-cancer	69.0±1.5	65.0±1.4 •	69.8±1.2	67.6±1.1	72.5±1.1 ◦	74.4±1.2 ◦	66.6±1.4 •
breast-w	95.2±0.7	95.1±0.6	95.0±0.7	95.2±0.6	95.7±0.3	96.0±0.3 ◦	95.6±0.3
german	70.3±0.7	70.4±0.7	70.4±1.1	70.5±0.5	70.5±0.8	70.9±0.8	67.2±1.2 •
glass (G2)	70.5±4.3	70.6±2.5	70.5±3.3	71.3±1.7	67.3±2.5	79.7±1.4 ◦	79.5±1.6 ◦
glass	59.8±1.4	59.3±1.4	59.6±1.1	60.2±1.3	60.1±1.6	59.9±2.1	59.3±1.4
heart-c	78.2±1.1	76.8±1.4	76.6±0.9 •	79.2±2.4	77.0±1.2	77.5±1.2	75.1±1.4 •
heart-h	73.9±0.9	72.6±1.6	74.8±1.2	73.7±0.9	77.8±1.2 ◦	79.5±0.8 ◦	76.6±1.0 ◦
heart-statlog	79.2±1.5	77.7±1.7 •	78.1±1.9 •	80.1±0.7	76.2±1.6 •	78.5±1.9	75.7±2.0 •
hepatitis	79.8±2.4	79.5±2.2	79.5±1.7	80.7±1.6	84.4±1.8 ◦	84.4±1.3 ◦	80.7±1.4
hypothyroid	91.7±0.1	91.7±0.0	91.7±0.0	91.9±0.0 ◦	91.7±0.0	91.9±0.0 ◦	91.7±0.1
ionosphere	87.0±1.0	86.7±0.8	87.4±0.8	88.1±0.5 ◦	87.8±1.4	87.2±0.6	86.6±0.7
iris	91.8±0.3	91.5±0.9	91.8±0.3	91.5±0.8	91.9±0.2	91.5±0.9	90.7±1.1
kr-vs-kp	99.3±0.1	99.3±0.1	99.3±0.1	99.4±0.1 ◦	99.3±0.1	99.5±0.1 ◦	99.5±0.1 ◦
lymphography	75.2±0.8	76.3±2.1	75.2±1.5	76.0±2.4	76.1±1.6	78.6±1.6 ◦	75.8±2.0
mushroom	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
pima-indians	74.0±0.8	72.9±0.7	74.2±0.5	71.9±0.4 •	74.1±0.6	74.1±0.5	69.4±0.8 •
primary-tumor	39.8±1.1	36.1±1.4 •	37.6±1.4 •	35.7±1.4 •	38.7±1.9	40.0±0.5	40.3±1.1
segment	91.0±0.2	91.2±0.3	91.1±0.2 ◦	91.3±0.2	91.5±0.3 ◦	91.8±0.2 ◦	91.8±0.3 ◦
sick	93.3±0.1	93.3±0.1	93.2±0.1 •	93.4±0.0 ◦	93.3±0.1	93.4±0.0 ◦	93.2±0.1 •
sonar	68.8±2.5	68.3±2.5	68.6±3.5	69.1±2.4	70.3±2.6	71.5±2.2	70.5±3.1
soybean	75.1±0.8	72.2±0.8 •	76.1±0.7 ◦	73.5±0.6 •	77.6±0.5 ◦	77.7±0.5 ◦	76.7±0.7 ◦
splice	92.6±0.3	92.3±0.3 •	92.2±0.3 •	93.4±0.2 ◦	93.2±0.2 ◦	94.2±0.2 ◦	92.2±0.2 •
vehicle	63.4±0.9	62.0±0.6 •	64.1±1.0 ◦	64.2±0.7	65.7±0.7 ◦	66.1±0.5 ◦	64.2±0.7
vote	95.4±0.4	95.5±0.4	95.5±0.3	95.6±0.5	95.5±0.4	95.5±0.4	96.2±0.5 ◦
vowel	77.9±1.0	78.0±1.0	79.5±1.0 ◦	80.8±1.0 ◦	73.8±0.6 •	76.6±0.5 •	78.2±0.7
zoo	92.5±1.8	92.8±1.6	94.0±2.0	94.8±2.1 ◦	89.6±1.4 •	90.8±1.5	91.5±1.4

Multistrategy Learning for Information Extraction

Dayne Freitag

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
dayne@cs.cmu.edu

Abstract

Information extraction (IE) is the problem of filling out pre-defined structured summaries from text documents. We are interested in performing IE in non-traditional domains, where much of the text is often ungrammatical, such as electronic bulletin board posts and Web pages. We suggest that the best approach is one that takes into account many different kinds of information, and argue for the suitability of a multistrategy approach. We describe learners for IE drawn from three separate machine learning paradigms: rote memorization, term-space text classification, and relational rule induction. By building regression models mapping from learner confidence to probability of correctness and combining probabilities appropriately, it is possible to improve extraction accuracy over that achieved by any individual learner. We describe three different multistrategy approaches. Experiments on two IE domains, a collection of electronic seminar announcements from a university computer science department and a set of newswire articles describing corporate acquisitions from the Reuters collection, demonstrate the effectiveness of all three approaches.

1 INTRODUCTION

Information extraction (IE) poses the following problem: Suppose each document in a collection describes some entity or event drawn from a semantically coherent domain. For example, the collection may consist of newswire articles describing terrorist attacks in Latin

America, or of personal home pages from a university computer science departments. Given a document from the collection and a set of questions defined for the domain, find the answer to each question in the form of a fragment of text from the document. In the case of articles on terrorism, the object might be to find the title of the group responsible for the attack, the instrument of the attack, and the victim's name; from home pages, we might seek to extract the owner's name, home address, and university affiliation.

There are many possible uses for a successful IE system. As a front end, an IE system can enable database mining and knowledge discovery in textual domains, where such processing would otherwise be limited or impossible. In hypertext, it can support directed and efficient automatic navigation. It can serve as a source of high-quality features for document categorization. And the output of an IE system can be viewed as a kind of succinct and directed summarization.

Although traditional IE (Cowie & Lehnert, 1996) concentrates on domains consisting of grammatical prose, we are interested in extracting information from "messy" text, such as Web pages, email, and finger plan files. Our goal is the development of machine learning methods for such domains. To perform well, these methods must be prepared to exploit non-linguistic information, such as stock phrases, document formatting, meta-textual structure (e.g., in HTML), and term frequency statistics.

Several learning IE systems have been proposed which are also targeted at such domains (Soderland, 1997) (Califf & Mooney, 1997) (Kushmerick, 1997). These previous investigations all take a single approach or attack a particular kind of domain. However, given the wealth of information in a typical document and the difficulty of adequately representing this information for learning, we surmise that no individual learn-

ing approach is best for all IE problems. An individual learner embodies biases that make it more suitable for some kinds of information and aspects of a problem than for others. A statistical learner like Naive Bayes, for example, is useful for problems in which each feature contributes some evidence toward the determination of class membership, and in which violations of the independence assumption do not predominate. It is less suitable for problems involving elaborate feature sets, in which some features are abstractions or combinations of others (i.e., where the independence assumption is directly violated). Symbolic learners, on the other hand, work quite well for problems with elaborate feature sets, especially for those classes expressible in logical terms using a small subset of features. These considerations suggest a *multistrategy* approach.

Multistrategy learning is an attempt to devise systems which, by employing multiple constituent learners, which are typically drawn from diverse paradigms, achieve performance superior to any single learner (Michalski & Tecuci, 1994). The bulk of emphasis in past research in this area has been on systems which combine analytical and empirical techniques. Our work, however, is an example of what has been called “empirical multistrategy learning” (Domingos, 1996). All constituent learners are inductive, each designed to solve the IE problem individually. Elsewhere we have shown that heuristic combination of two learners from different paradigms can yield substantial performance improvements for the IE problem (Freitag, 1997). Here, we ask how we might profitably combine component learners by treating them as black boxes. This approach has been called “meta-learning” in the literature (Chan & Stolfo, 1993). Although we might expect a heuristic combination to achieve better performance, there are clear advantages to the meta-learning approach. It is modular and flexible, making no assumptions about the design of component learners or the number of learners available.

In this paper, we introduce three machine learning algorithms for IE, each drawn from a different paradigm and each suitable for particular kinds of IE problems. Next, we describe three ways of combining the basic learners, all variations of the meta-learning idea. Finally, we describe a set of experiments on two IE domains.

2 LEARNING TO EXTRACT

In the simplest version of the information extraction problem, a single set of questions is applied to each document in a domain, and a single text fragment is sought as the answer to each question. We call a single question a *field*; the answer fragment from an individual document is a *field instance* or *instantiation*. For example, in a domain consisting of newswire articles describing terrorist attacks, one field might be the *perpetrator* of the attack, and the instantiation of this field in a given article might be “FMLN.”

A field can be formalized as a function $\mathcal{F}(D) = (b_b, b_e)$ that maps a document to the boundaries of a text fragment (b_b and b_e are the indexes of the beginning and ending boundary terms, respectively). Given a set of documents in which this mapping is labeled, the goal of a ML system is to learn the function $\hat{\mathcal{F}}$ that best approximates \mathcal{F} . This can be realized in the form of an auxiliary function $\mathcal{G}(D, b_b, b_e) = \mathbf{R} \cup \{\text{nil}\}$, which, given a candidate fragment, either returns a confidence that it is a field instance or declines to issue a confidence (*nil*). The form of \mathcal{G} has a convenient affinity with any number of ML algorithms (the *nil* in its range constitutes a failure to match, for algorithms that include a notion of matching). The three approaches we will discuss, all based on standard ideas from ML, each implement \mathcal{G} .

Note that this learning task is only a part of the functionality of a typical participating system at the Message Understanding Conference (MUC) (Cardie, 1997). What we have called *fields* correspond to *slots* in the MUC setting. A slot is a component of a larger structure, called a *template*, which summarizes the relevant information contained in a document. In addition to the slot-filling task, which we address here, the more general MUC problem includes tasks such as document relevance determination, discourse analysis, and template merging. Thus, our results are best regarded as a piece of the larger IE puzzle.

2.1 ROTE LEARNING

Perhaps the simplest possible learning approach to the IE problem is to memorize field instances verbatim. Presented with a novel document, this memorizing learner simply matches text fragments against its “learned” dictionary, saying “field instance” to any matching fragments and rejecting all others.

As a slightly more sophisticated approach, we can estimate the probability that the matched fragment is

indeed a field instance. The dictionary learner we experiment with here, which we call *Rote*, does exactly this. Training *Rote* involves scanning the training corpus and storing all distinct field instances verbatim in its dictionary. Dictionary construction is followed by a second pass through the training corpus. For each text fragment in its dictionary, *Rote* counts the number of times it appears as a field instance (*pos*) and the number of times it occurs over all (*tot*). During test, *Rote*'s confidence in a prediction is the value $(pos + 1)/(tot + 2)$, i.e., a Laplace estimate that the matching fragment is genuine.

This approach, simple as it is, is nevertheless surprisingly applicable in a wide variety of domains. Its confidence, moreover, correlates well with actual probability of correctness. Because of this, even low-confidence predictions are potentially useful.

2.2 TERM-SPACE LEARNING

It is straightforward to adapt ideas from document classification to the IE setting. A simple mapping might transform every field instance into a miniature "document" and apply "bag-of-words" algorithms directly, such as Rocchio with TFIDF term weighting or Naive Bayes. Such an approach could be viewed as a generalization of *Rote*.

In contrast with document classification, however, positive examples in an IE setting always occur embedded within some larger context. This context is often critical in disambiguating field instances from other fragments. Although it is hard to exploit contextual regularities by memorizing, statistical approaches are well suited for this.

We base our bag-of-words learner, which we call *Bayes*, on the Naive Bayes algorithm, as used in document classification and elsewhere (originally in (Maron, 1961)). Each fragment of text in a document (of appropriate size) is regarded as a competing hypothesis. Given a document, we want to find the most likely hypothesis (the fragment most likely to be a field instance). Bayes Rule tells us how to maintain our belief in a set of disjoint hypotheses (H_i) in reaction to observed data (D):

$$\Pr(H_i|D) = \frac{\Pr(D|H_i) \Pr(H_i)}{\sum_{j=1}^n \Pr(D|H_j) \Pr(H_j)}$$

As in Naive Bayes as used elsewhere, the important terms to estimate are $\Pr(H_i)$ (the *prior* probability) and $\Pr(D|H_i)$ (the conditional *data* probability).

We assume a hypothesis takes the form, "the field instance starts at token s and is k tokens long" (let $H_{s,k}$ represent such a hypothesis). In other words, a single hypothesis consists of two parts, *position* and *length*. We can estimate the probability of a particular position or length from training data. In our implementation we treat these two estimates as independent, which is different from the typical Naive Bayes data independence assumption, but similar in spirit. Thus, our prior $\Pr(H_{s,k})$ is simply the product of $\Pr(\text{position} = s)$ and $\Pr(\text{length} = k)$.

Bayes's data likelihood estimate, $\Pr(D|H_{s,k})$, is based on the terms that occur in and around the text fragment to which $H_{s,k}$ corresponds. This estimate is formed in a way similar to Naive Bayes for document classification (a product of individual term estimates), but with a few modifications for the IE setting. In particular, a context window parameter w is set prior to training, and the w tokens on either side of a fragment are used to form the estimate, in addition to the in-field tokens. The algorithm is described in greater detail elsewhere (Freitag, 1997).

2.3 RELATIONAL LEARNING

Both *Bayes* and *Rote* are hobbled by their inability to take into account anything but simple term frequency statistics. It may be the case, however, that the information needed to perform information extraction comes in other forms. More abstract clues may be important, such as linguistic syntax, document layout, or simple orthography. In addition, statistical approaches like *Bayes* work by summing all available evidence, whereas in IE a more fruitful approach may involve identifying simple patterns that serve to distinguish sub-classes of a field.

Symbolic learning algorithms from the "covering" family form hypotheses that match such data spaces well. Previous research has shown the effectiveness of such methods for the IE problem (Soderland, 1996) (Califf & Mooney, 1997). Our relational learner, called *SRV*, is a variant of FOIL (Quinlan, 1990). Its example space consists of all text fragments from the training document collection as long (in number of tokens) as the smallest field instance in the training corpus but no longer than the largest. A negative example is any fragment that is not tagged as a field instance. Note that this includes fragments that contain, are contained by, and overlap with field instances.

Induction proceeds as with FOIL: Starting with a null rule that matches all examples not covered by previ-

ously learned rules, SRV greedily adds predicates using FOIL's information gain metric. In addition to the tagged document collection, SRV takes as input a set of features to use in conducting search. These features come in two varieties, *simple* features, which map from an individual token to an arbitrary value (e.g., capitalized? or noun?), and *relational* features, which map from a token to another token (e.g., next-token or subject-verb).

An individual predicate in SRV belongs to one of a few predefined types:

- **length(Relop N):** The number of tokens in a fragment is less than, greater than, or equal to some integer.
- **some(Var Path Feat Value):** This is a feature-value test for some token in the sequence (e.g., "the fragment contains some token that is capitalized"). One argument to this predicate is a variable. For a rule to match a text fragment, each distinct variable in a rule (used in this or either of the position predicates below) must bind to a distinct token in the fragment.
- **every(Feat Value):** Every token in a fragment passes some feature-value test (e.g., "every token in the fragment is non-numeric").
- **position(Var From Relop N):** This constrains the position of a token bound by a *some*-predicate in the current rule. The position is specified relative to the beginning or end of the sequence.
- **relpos(Var1 Var2 Relop N):** This constrains the ordering and distance between two tokens bound by distinct variables in the current rule.

Relational features are used only in the *Path* argument to the *some* predicate. This argument can be empty, in which case the *some* predicate is asserting a feature-value test for a token actually occurring within a field, or it can be a list of relational features. In the latter case, it is positing both a relationship about a field token with some other nearby token, as well as a feature-value for the other token. For example, the assertion:

```
some(?A [prev-token prev-token] capitalized true)
```

amounts to the English statement, "There is some token preceded by a capitalized token two tokens back."

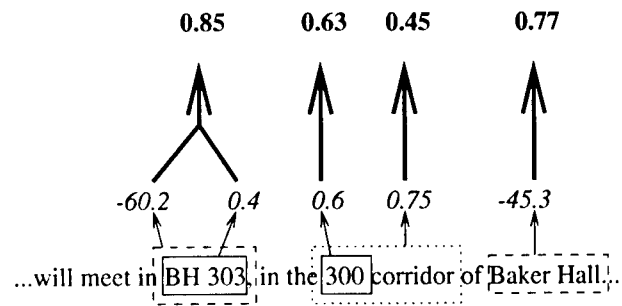


Figure 1: Hypothetical Extraction of a Seminar Location. Each box style is intended to represent a different learner. By combining evidence from multiple learners, we can correct for the mistakes of individual learners.

In order to enable SRV to return confidences with its predictions, training is followed by a validation step. Rather than train on the entire training collection, we set aside a fraction of the documents (one-third here) for validation. With each rule learned by SRV we store its performance on the hold-out set. From this performance we estimate a rule's actual accuracy. The confidence of a prediction made by SRV is formed from the estimated accuracy of matching rules. For additional details on SRV, please refer to (Freitag, 1998).

3 COMBINING LEARNERS

Certain features of the IE problem make it particularly amenable to a multistrategy approach. Among these are the following:

- **Examples have multiple representations.** Because documents and text fragments are "natural" objects which must be mapped to appropriate representations for learning, multiple mappings are possible. Although some information is necessarily lost in any one mapping, we can hope that taking multiple views of a document will permit better overall performance.
- **The problem is essentially Boolean.** As outlined above, performing extraction can be reduced to the task of accepting or rejecting candidate text fragments. Consequently, we can gauge a learner's performance on validation documents in an attempt to model the relationship between prediction confidence and probability of correctness.
- **Each document is a case study.** In contrast with a traditional classification problem, each per-

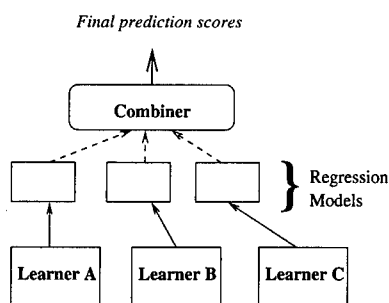


Figure 2: The Basic Combination Scheme. Regression models based on learner performance on hold-out sets are used to map raw confidence scores to probabilities. The combiner uses these probabilities to order all predictions.

formance unit, a document, is a collection of test problems. *Overgeneration*, the problem of saying *yes* to too many text fragments, can be regarded as an asset when multiple learners are available. It both affords more data for our attempt to model a learner's usefulness, and holds forward the hope that the poor predictions of a single learner can be corrected by checking them against those of other learners.

Figure 1 shows a hypothetical excerpt from a seminar announcement and how such correction might take place.

3.1 BASIC COMBINATION METHOD

Within the constraint that all learners assign a confidence to any predictions they make (any fragments they accept), a wide range of behaviors is possible. In particular, for a number of reasons, we cannot assume that the confidences bear any resemblance to true probability of correctness, or even that they are comparable across learners. Bayes's confidences are large negative log probabilities, for example.

We *do* assume, however, that probability of correctness increases with increasing confidence for all learners. The basic idea, therefore, is to attempt to compute a mapping for each learner from confidence to probability of correctness. Figure 2 shows this in outline. The specific steps involved are:

1. **Validate performance on a hold-out set.** Reserve a part of the training set for validation. After training each learner, store its predictions, with confidences, on the hold-out set.

2. **Use regression to map confidences to probabilities.** Based on the learner's performance on the hold-out set, attempt to model how its performance varies with confidence. What is modeled, and the kind of regression used, depends on the combination method.
3. **Use the regression models and calculated probabilities to make the best choice on the test set.**

We experimented with three basic methods of combination. The first two, which we will call **Max** and **Prob**, both attempt to work with regression models that map directly from confidence to probability of correctness. The third, which we will call **CBayes**, uses Bayes Rule to make combination decisions.

3.2 REGRESSION TO ESTIMATE CORRECTNESS

If a learner's confidence numbers are meaningful, then the probability that a prediction is correct will increase with increasing confidence. We use linear regression to model the rate at which this probability increases. For each prediction made we create a datapoint (x, y) , where x is the prediction confidence, and y is 1, if the prediction was correct (the corresponding fragment was a field instance), else 0.

The result is a line equation which we use directly to map from learner confidence to probability of success. Both **Max** and **Prob** use the resulting estimates to arbitrate among multiple learners' predictions for a document. Estimates are computed for each learner's predictions, and the prediction with the highest estimate is chosen as the top combined prediction. The two methods differ only in how they handle the case in which multiple learners offer predictions for the same text fragment. In such an event, **Max** simply takes the larger estimate as the probability that the fragment is a field instance.

We believe, however, that the fact that two or more learners agree on a prediction provides more information than either prediction alone. Indeed, if we assume that two probability estimates of an event, P_a and P_b , are independent, then the combined probability is the probability that they are not both wrong, i.e., $1 - (1 - P_a)(1 - P_b)$. **Prob's** estimate is based on this assumption. Given a set of probability estimates P_i , its estimate for the combined probability is $1 - \prod_i (1 - P_i)$.

3.3 BAYESIAN PREDICTION COMBINATION

Although Prob may exploit the availability of predictions from multiple learners better than Max, it still leaves something to be desired. In particular, it ignores some of the available information, such as the frequency with which a learner tends to predict at a given confidence level and any notion of prior probabilities.

For our final combination method, we attempt to apply Bayes Rule, which tells us how to maintain our probability estimates in response to incoming data. Using Bayes Rule offers two advantages over Prob: It allows us to incorporate priors into our estimates, and it tells us how to maintain our hypothesis space so that the resulting estimates are closer to true probabilities—an advantage in terms of the accuracy-coverage trade-off.

Here, a hypothesis H_i takes the form, “*the fragment at this place in the document is a field instance.*” Let $P_{ai} = C$ be the event, *Learner A predicted fragment i is a field instance with confidence C* . For each fragment i chosen by any of the learners, we maintain two hypotheses explicitly, H_i and $\neg H_i$. Individual learner predictions $P_{ai} = C$ are treated as events which cause us to update hypotheses. We want, therefore, to model $\Pr(P_{ai} = C|H_i)$ and $\Pr(P_{ai} = C|\neg H_i)$. It is more convenient, however, to model the event $P_{ai} \geq C$, i.e., the probability of a prediction with confidence *at least* C . Modeling the cumulative probability yields better statistics and allows us to avoid the arbitrary decisions inherent in binning.

We use exponential regression to model these two probabilities, i.e., we perform linear regression on pairs of the form $(x, \log(y))$, where x is a confidence level, and y is the cumulative probability of seeing a prediction for a fragment given that it either is or is not a field instance. As an example, consider the problem of creating the “positive” model $\Pr(P_{ai} \geq C|H_i)$ for some learner A . Let F be the total number of field instances in the validation set, and let $G_a(C)$ be the number of field instances identified by Learner A with predictions having confidence equal to or greater than C . For every prediction made by Learner A , we add a regression datapoint $(x, \log(y))$, where x is the confidence of the prediction and $y = G_a(x)/F$. The “negative” model $\Pr(P_{ai} \geq c|\neg H_i)$ is constructed in the same way, except over non-field-instance fragments—any fragment in the validation set identified by any of the learners. We settled on exponential regression empirically, but it is easy to see why it works better than

Table 1: Accuracy-Coverage Results for the Seminar Announcement Domain.

	speaker		location	
	Acc	Cov	Acc	Cov
Rote	57.4 ± 8.8	11.8	89.5 ± 2.2	64.9
Bayes	36.1 ± 3.5	70.8	59.6 ± 2.8	98.7
SRV	60.4 ± 3.0	96.6	75.9 ± 2.6	92.3
Max	59.8 ± 3.0	98.8	75.6 ± 2.5	99.7
Prob	60.8 ± 3.0	98.8	76.0 ± 2.5	99.7
CBayes	62.5 ± 3.0	98.8	75.6 ± 2.5	99.7
	stime		etime	
	Acc	Cov	Acc	Cov
Rote	73.7 ± 2.5	99.6	75.1 ± 3.7	95.4
Bayes	98.2 ± 0.7	100.0	96.1 ± 1.6	99.6
SRV	98.6 ± 0.7	99.8	94.1 ± 2.0	98.4
Max	96.6 ± 1.0	100.0	93.6 ± 2.0	100.0
Prob	99.3 ± 0.5	100.0	95.4 ± 1.7	100.0
CBayes	99.3 ± 0.5	100.0	96.3 ± 1.6	100.0

linear regression. Low-confidence predictions tend to be more frequent than high-confidence ones, obeying something like Zipf’s Law.

With each prediction, we use the two models associated with a learner to adjust the posterior probabilities of the two mutually exclusive hypotheses regarding the affected fragment, always normalizing so they sum to 1.

4 EXPERIMENTS

We experimented with data from two IE domains. One consists of 485 postings to electronic bulletin boards, which describe upcoming seminars in a university environment. The earliest of these announcements dates to October, 1982; the most recent was posted in August, 1995. We manually tagged these announcements for four fields: **speaker**, **location**, **stime** (start time), and **etime** (end time). The other domain is a collection of 600 newswire articles on corporate acquisitions from the Reuters data set (Lewis, 1992). We defined nine fields for this domain and manually annotated the collection to identify all instances of them. We selected five of the fields for these experiments: **acquired** (the official name of the company or resource that is being purchased), **purchaser**, **acqabr** (the short name for **acquired** used in the body of the article), **purchabr**, and **dlrmt** (the price paid).

The performance numbers we report here are the result of five-fold experiments in each domain. In each iteration the datasets were randomly divided into two partitions of equal size. One partition was used for training, the other for testing.

Table 2: Accuracy-Coverage Results for the Acquisition Domain.

	acquired		purchaser	
	Acc	Cov	Acc	Cov
Rote	56.1 ± 5.6	20.5	47.5 ± 5.6	22.3
Bayes	22.4 ± 2.2	96.4	41.4 ± 2.6	99.7
SRV	41.1 ± 2.6	96.0	49.7 ± 2.7	97.8
Max	43.4 ± 2.5	99.8	51.4 ± 2.7	99.9
Prob	45.0 ± 2.5	99.8	53.2 ± 2.7	99.9
CBayes	45.8 ± 2.5	99.8	54.7 ± 2.6	99.9
	acqabr		purchabr	
	Acc	Cov	Acc	Cov
Rote	31.7 ± 4.2	43.8	24.7 ± 4.1	38.5
Bayes	33.1 ± 2.8	99.7	52.0 ± 2.9	99.9
SRV	45.0 ± 3.0	99.8	54.0 ± 2.9	99.6
Max	42.7 ± 2.9	100.0	57.4 ± 2.9	100.0
Prob	47.6 ± 3.0	100.0	61.0 ± 2.9	100.0
CBayes	47.2 ± 3.0	100.0	60.0 ± 2.9	100.0
	dlramt			
	Acc	Cov		
Rote	77.2 ± 4.7	48.1		
Bayes	62.2 ± 4.3	76.9		
SRV	74.4 ± 3.5	90.1		
Max	72.0 ± 3.5	95.5		
Prob	73.1 ± 3.5	95.5		
CBayes	70.2 ± 3.5	95.5		

Table 3: F1 Scores. Two scores are shown for each result: *Full*, the F1 score for the accuracy-coverage results reported in Tables 1 and 2, and *Peak*, the highest F1 score along the full accuracy-coverage curve.

	speaker		location		stime	
	Full	Peak	Full	Peak	Full	Peak
Rote	19.6	19.6	75.3	75.3	84.7	84.7
Bayes	47.8	48.0	74.3	75.1	99.1	99.1
SRV	74.3	74.3	83.3	83.3	99.2	99.2
Max	74.5	74.5	86.0	86.4	98.3	98.3
Prob	75.3	75.3	86.3	86.6	99.6	99.6
CBayes	76.6	76.6	86.0	86.0	99.6	99.6
	etime		acquired		purchaser	
	Full	Peak	Full	Peak	Full	Peak
Rote	84.0	84.0	30.0	30.0	30.3	30.3
Bayes	97.8	97.8	36.4	38.3	58.5	59.3
SRV	96.2	96.2	57.5	58.3	65.9	66.4
Max	96.7	96.7	60.5	61.2	67.9	68.0
Prob	97.6	97.6	62.0	62.7	69.5	69.7
CBayes	98.1	98.1	62.8	63.2	70.7	71.1
	acqabr		purchabr		dlramt	
	Full	Peak	Full	Peak	Full	Peak
Rote	36.8	37.2	30.1	30.8	59.3	59.3
Bayes	49.7	52.8	68.4	68.6	68.8	68.8
SRV	62.0	62.0	70.0	70.2	81.5	81.5
Max	59.8	59.8	72.9	72.9	82.1	82.1
Prob	64.5	64.5	75.8	75.8	82.8	82.8
CBayes	64.1	64.1	75.0	75.0	80.9	80.9

A third of the training set, randomly selected, was set aside for validation. Each learner was trained on the remaining two-thirds, and tested on the validation set. Following this validation step, each learner was again trained on the entire training set and tested on the test set. The goal of the combining methods was to use performance results on the validation set to arbitrate among predictions on the test set.

The performance of all methods is summarized in Table 1, for the seminar announcement fields, and Table 2, for the acquisition fields. The unit of measurement here, as elsewhere in this paper, is a document. When assessing a learner's performance for a single document, we can distinguish among four basic outcomes: no prediction from the learner, prediction on a document lacking a field instance (*spurious*), top prediction is incorrect (*wrong*), and top prediction is correct (*correct*). The coverage column (*Cov*) shows for what fraction of those documents containing a field instance a learner actually made a prediction. The number in the accuracy column (*Acc*) shows the fraction of correct predictions over documents for which the learner made a prediction *and* which contained a field instance, i.e., it ignores spurious predictions. Note that if any single learner makes a spurious prediction, all combining methods also make one, since they are limited to ordering the predictions made by actual learners. Thus, counting spurious predictions as errors, while generally appropriate, tends to obscure the differences between the learners and the combining methods.

Both the accuracy and coverage values should be considered together. There are cases, for example, where the accuracy number makes Rote look like the strongest extraction method. Its accuracy, however, is usually measured over a much smaller number of documents. While it can typically recognize a fraction of field instances with reasonable accuracy (especially locations), it does not stand up well to overall comparison with the other learners. For convenience in comparing systems, it is common in information retrieval and information extraction to combine precision and recall into a single, summary number, called the *F-measure*:

$$F = \frac{(\beta^2 + 1.0)PR}{(\beta^2 P) + R}$$

The parameter β determines how much to favor recall over precision. Researchers in information extraction frequently report the F1 score of a system ($\beta = 1$), which weights precision and recall equally. We can do the same with our accuracy-coverage results. Table 3

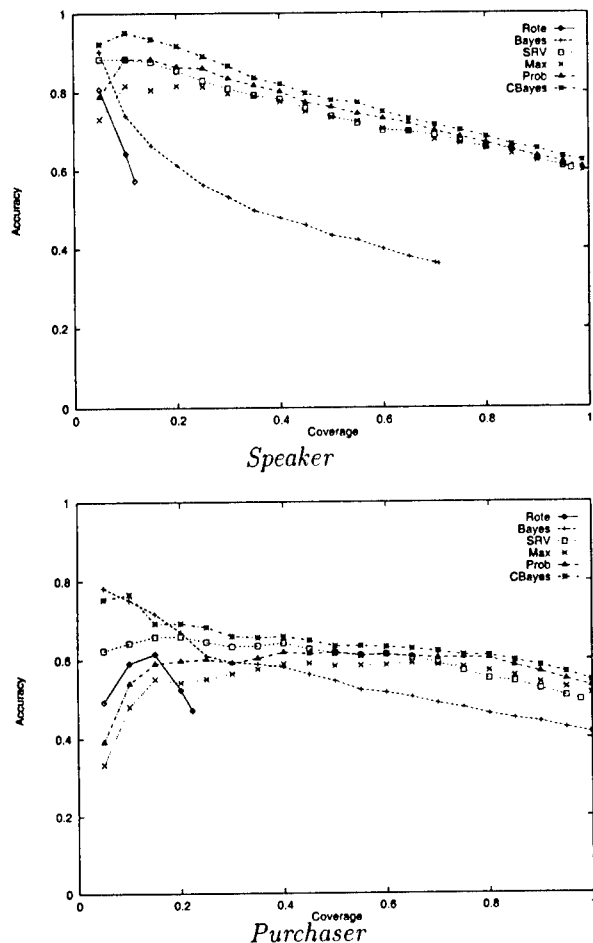


Figure 3: Plots of accuracy vs. coverage for all methods on two fields, **speaker** and **purchaser**.

shows the F1 scores for all learners and fields.

For the **purchabr** field there is clear statistical separation between the best individual learner (SRV) and the top two combining methods (Prob and CBayes). Note, as Table 3 makes clear, that even in the cases where the difference is less apparent, the combining methods tend to outperform the best individual method at *higher* coverage levels. Among the three combining methods there is not one case of statistical separation, but across all fields a clear picture emerges in which Prob and CBayes are better than Max. Note that even in cases where a combining method performs only as well as the best individual learner, it has served a valuable purpose—that of relieving us of the requirement of choosing a single learner. If a combining method can do this in most cases, while providing added value in a few, we account it a clear success.

Perhaps more interesting than summary statistics are

Table 4: Overlap in Learner Behavior for the **Speaker** Field. Numbers are the probability that column learner predicted correctly, given that the row learner predicted correctly.

	Rote	Bayes	SRV	Max	Prob	CBayes
Rote	1	0.81	0.81	0.90	0.96	0.97
Bayes	0.22	1	0.68	0.86	0.89	0.86
SRV	0.09	0.30	1	0.93	0.93	0.97
Max	0.10	0.37	0.92	1	0.99	0.98
Prob	0.11	0.38	0.90	0.98	1	0.98
CBayes	0.11	0.35	0.92	0.93	0.95	1

accuracy-coverage (similar to *precision-recall*) graphs. Each point x along the horizontal axis represents the $x\%$ most confident predictions. The vertical value at this point is the accuracy of these predictions. If the accuracy-coverage curve declines monotonically, it suggests that the learner's confidence correlates well with actual accuracy.

Figure 3 shows the accuracy-coverage curves for all methods on two of the fields. The **speaker** and **purchaser** fields are the ones for which CBayes does best. These graphs make clear what the summary statistics cannot: That combining learners allows us to make better accuracy-coverage judgments than we can with a single learner. The anomalous high-confidence behavior of Prob and Max in the **purchaser** curve may be due to an over-reliance on Rote, which has similar behavior. Note that the high-confidence (low-coverage) end of the curve is the part with the least statistical certainty. Also, although CBayes appears better than any *individual* learner, an examination of the graphs for all fields does not support a preference of it over Prob, or vice versa. There are cases where CBayes has high-confidence difficulties similar to those shown here for Prob and Max. We believe that better regression models will mitigate some of these phenomena.

The strength of a meta-learning approach depends on the mutual independence of the constituent learners. Table 4 shows where some of the power of combining learners comes from on the **speaker** field, a relatively challenging task. In this table we ask the question, given that Learner A has predicted correctly on some document, what is the probability that Learner B will also predict correctly? The number in entry (i, j) is the fraction of all documents correctly handled by method i which method j also correctly handled. Based on this table, it is evident that Rote and Bayes are more closely related to each other than either to SRV.

The column for a combining method allows us to infer which learners it depends on most for its performance. It appears from this that all three methods rely more on *Rote* than on *Bayes*. We would hope to see this, based on Figure 3, since the few *Rote* predictions that are available for this field tend to have higher accuracy than most *Bayes* predictions. It is also gratifying that all methods appear to rely heavily on *SRV*, since it is the best individual learner in this case.

5 CONCLUSION

The experimental results presented here show that multistrategy learning can be useful for the problem of information extraction. We present one form of multistrategy learning, in which the component learners are treated as black boxes and only their reliability, as a function of confidence, is modeled. Nothing in the basic framework requires the information extraction setting or makes any assumptions about the number or structure of component learners. It is only necessary that learners be instrumented to associate a confidence with any prediction they make, something which is already part of the design of many learners, and which can be readily added to others.

We do not claim that the multistrategy results reported here are the best that can be achieved. Many details remain to be filled in, such as how best to conduct validation and which statistical assumptions are appropriate. We have experimented with two kinds of regression to model learner reliability, but would not be surprised if other methods which we have not tried, such as logistic regression or a simple neural network, might afford increased accuracy. We regard this as future work.

It also remains to be seen how these results might be fit into a more traditional information extraction setting, in which slot filling is performed as part of a larger system and as one of several interacting tasks. Still, the approaches described here are immediately applicable to a number of unconventional information extraction problems. And we can begin to see how information extraction from ungrammatical text, and other "natural" problems admitting multiple abstract representations, can be addressed with machine learning methods.

Acknowledgements

This research was supported in part by the DARPA HPKB program under contract F30602-97-1-0215.

References

- Califf, M. E., and Mooney, R. J. 1997. Relational learning of pattern-match rules for information extraction. In *Working Papers of ACL-97 Workshop on Natural Language Learning*.
- Cardie, C. 1997. Empirical methods in information extraction. *AI Magazine* 18(4):65-79.
- Chan, P., and Stolfo, S. 1993. Experiments on multistrategy learning by meta-learning. In *Proceedings of the Second International Conference on Information and Knowledge Management (CIKM 93)*, 314-323.
- Cowie, J., and Lehnert, W. 1996. Information extraction. *Communications of the ACM* 39(1).
- Domingos, P. 1996. Unifying instance-based and rule-based induction. *Machine Learning* 24(2):141-168.
- Freitag, D. 1997. Using grammatical inference to improve precision in information extraction. In *Notes of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*.
- Freitag, D. 1998. Information extraction from html: Application of a general machine learning approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.
- Kushmerick, N. 1997. *Wrapper Induction for Information Extraction*. Ph.D. Dissertation, University of Washington. Tech Report UW-CSE-97-11-04.
- Lewis, D. 1992. *Representation and Learning in Information Retrieval*. Ph.D. Dissertation, Univ. of Massachusetts. CS Tech. Report 91-93.
- Maron, M. 1961. Automatic indexing: An experimental inquiry. *Journal of the Association for Computing Machinery* 8:404-417.
- Michalski, R., and Tecuci, G., eds. 1994. *Machine Learning: A Multistrategy Approach*. San Mateo, CA: Morgan Kaufmann.
- Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239-266.
- Soderland, S. 1996. *Learning Text Analysis Rules for Domain-specific Natural Language Processing*. Ph.D. Dissertation, University of Massachusetts. CS Tech. Report 96-087.
- Soderland, S. 1997. Learning to extract text-based information from the world wide web. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*.

An Efficient Boosting Algorithm for Combining Preferences

Yoav Freund

AT&T Labs

yoav@research.att.com

Raj Iyer*

MIT Laboratory for Computer Science

rajiyer@mit.edu

Robert E. Schapire

AT&T Labs

schapire@research.att.com

Yoram Singer

AT&T Labs

singer@research.att.com

Abstract. The problem of combining preferences arises in several applications, such as combining the results of different search engines. This work describes an efficient algorithm for combining multiple preferences. We first give a formal framework for the problem. We then describe and analyze a new boosting algorithm for combining preferences called RankBoost. We also describe an efficient implementation of the algorithm for a restricted case. We discuss two experiments we carried out to assess the performance of RankBoost. In the first experiment, we used the algorithm to combine different WWW search strategies, each of which is a query expansion for a given domain. For this task, we compare the performance of RankBoost to the individual search strategies. The second experiment is a collaborative-filtering task for making movie recommendations. Here, we present results comparing RankBoost to nearest-neighbor and regression algorithms.

1 Introduction

Consider the following movie-recommendation task, sometimes called a “collaborative-filtering” problem [8, 14]. In this task, a new user, Alice, seeks recommendations of movies that she is likely to enjoy. A collaborative-filtering system first asks Alice to rank movies that she has already seen. The system then examines the rankings of movies provided by other viewers and uses this information to return to Alice a list of recommended movies. To do that, the recommendation system looks for users whose preferences are similar to those of Alice and combines their recommendations.

One important property of this problem is that the most relevant information to be combined represents *relative preferences* rather than *absolute ratings*. In other words, even if the ranking of movies is expressed by assigning each movie a numeric score, we would like to ignore the absolute values of these scores and concentrate only on their relative order. This distinction becomes very important when we combine the rankings of many users who often use completely different ranges of scores to express identical preferences. Situations where we need to combine the ranking of different models also arise in meta-searching problems [5] and in information-retrieval problems [11, 10].

In this paper, we introduce and analyze an efficient algorithm called RankBoost for combining multiple rank-

ings. This algorithm is based on Freund and Schapire’s [6] AdaBoost algorithm and its recent successor developed by Schapire and Singer [13]. Similar to other boosting algorithms, RankBoost works by combining many “weak” rankings of the given instances. Each of these may be only weakly correlated with the target ranking that we are attempting to approximate. We show how to combine such weak rankings into a single highly accurate ranking, and we prove a bound on the quality of this final ranking in terms of the quality of the weak rankings.

For the movie task, we use very simple weak rankings which partition all movies into only two equivalence sets, those which are more preferred and those which are less preferred. For instance, we might use another user’s ranked list of movies partitioned according to whether or not he prefers them to some particular movie that appears on his list. Such partitions of the data have the advantage that they only depend on the relative ordering defined by the given rankings rather than absolute ratings. Despite their apparent weakness, their combination using RankBoost performs quite well experimentally.

Besides giving a theoretical analysis of the quality of the ranking produced by RankBoost, we also analyze its complexity and show how it can be implemented efficiently. We discuss further improvements in efficiency which are possible in certain natural cases.

We report the results of experimental tests of our approach on two different problems. The first is the meta-searching problem. In a meta-search application, the goal is to combine the rankings of several WWW search strategies. Each search strategy is an operation which takes as input a query, performs some simple transformation of the query (such as adding search directives such as “AND”, or search tokens such as “homepage”) and sends it to a particular search engine. The outcome of using each strategy is a list of URLs which are proposed as answers to the query. The goal is to combine the strategies that work best for a given set of queries.

The second problem is the movie-recommendation problem described above. For this problem, there exists a large publicly available dataset which contains ratings of movies by many different people. We compared RankBoost to nearest-neighbor and regression algorithms which have been previously studied for this application using several evaluation measures.

*Research conducted while visiting AT&T Labs and with support from an NSF Graduate Fellowship.

Despite the wide range of applications that use and combine rankings, this problem has received relatively little attention in the machine-learning community. The few methods that have been devised for combining rankings tend to be based either on nearest-neighbor methods [9, 14] or numerical-optimization techniques [1, 3]. In the latter case, the rankings are viewed as real-valued scores and the problem of combining different rankings reduces to numerical search for a set of parameters that will minimize the disparity between the combined scores and the feedback of a user.

While the above (and other) approaches might work well in practice, they still do not guarantee that the combined system will match the user's preference when we view the scores as a means to express preferences. Recently, Cohen, Schapire and Singer [4] proposed a framework for manipulating and combining multiple rankings in order to directly minimize the number of disagreements. In their framework, the rankings are used to construct preference graphs and the problem is reduced to a *combinatorial* optimization problem which turns out to be NP-complete; hence, an approximation is used to combine the different rankings. They also describe an efficient *on-line* algorithm for a related problem.

The algorithm we present in this paper uses a similar framework to theirs, but sidesteps the intractability problems. Furthermore, RankBoost is more appropriate for batch settings where there is "enough" time to find a good combination. Thus, the two approaches complement each other. Together, these algorithms constitute a viable approach to the problem of combining multiple rankings, that, as our experiments indicate, work very well in practice.

2 A formal model of the ranking problem

In this section, we describe our formal model for studying ranking. Let \mathcal{X} be a set called the *domain* or *instance space*. Elements of \mathcal{X} are called *instances*. For example, in the movie-ranking task, each movie is an instance.

A learning algorithm in our model accepts as input a set of *ranking features* f_1, \dots, f_n . These are intended to provide a base level of information about the ranking task. Said differently, the learner's job will be to learn a ranking expressible in terms of the ranking features, similar to ordinary features in more conventional learning settings. For the movie task, each ranking feature corresponds to a single viewer's past ratings of movies.

Formally, each ranking feature f_i is a function of the form $f_i : \mathcal{X} \rightarrow \mathbb{R}$. The set \mathbb{R} consists of all real numbers, plus one additional element ϕ which indicates that no ranking is given and which is defined to be incomparable to all real numbers. For two instances x_0 and x_1 , we interpret $f_i(x_1) > f_i(x_0)$ to mean that x_1 is ranked higher than x_0 by f_i . If $f_i(x) = \phi$ then x is unranked by f_i . For the movie ranking task, $f_i(x)$ is simply the numerical rating provided by movie-viewer i on movie x , or ϕ if the movie was not rated.

The final input to the learning algorithm is a *feedback function* Φ . This function encodes known relative ranking

information about a subset of the instances. Typically, the learner will try to approximate Φ to produce a ranking of unseen instances. For the movie task, the feedback consists of the known movie preferences provided by the current movie-viewer (i.e., the one for whom the system is currently attempting to recommend movies).

Formally, we assume the feedback function has the form $\Phi : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ with the interpretation that $\Phi(x_0, x_1)$ represents the degree to which x_1 should be correctly ranked above x_0 . Large positive values mean that x_1 should be ranked above x_0 while negative values mean the opposite; a value of zero indicates no preference between x_0 and x_1 . Consistent with this interpretation, we assume that $\Phi(x, x) = 0$ for all $x \in \mathcal{X}$, and that Φ is anti-symmetric in the sense that $\Phi(x_0, x_1) = -\Phi(x_1, x_0)$ for all $x_0, x_1 \in \mathcal{X}$. Note, however, that we do not assume transitivity of the feedback function.

For the movie task, we can define $\Phi(x_0, x_1)$ to be +1 if movie x_1 was preferred to movie x_0 by the current viewer, -1 if the opposite was the case, and 0 if either of the movies was not seen or if they were equally rated.

We generally assume that the support of Φ is finite. Let \mathcal{X}_Φ denote the set of *feedback instances*, i.e., those instances which occur in the support of Φ :

$$\mathcal{X}_\Phi = \{x \in \mathcal{X} \mid \exists x' \in \mathcal{X} : \Phi(x, x') \neq 0\}.$$

Also, let $|\Phi|$ be the size of the support of Φ :

$$|\Phi| = |\{(x_0, x_1) \in \mathcal{X} \times \mathcal{X} \mid \Phi(x_0, x_1) \neq 0\}|.$$

In some settings, it may be appropriate for the learner to accept a set of feedback functions Φ_1, \dots, Φ_m . However, all of these can be combined into a single function Φ simply by adding them: $\Phi = \sum_j \Phi_j$. (If some have greater importance than others, then a weighted sum can be used.)

Formally, we require the learner to output a ranking of all instances represented in the form of a function $H : \mathcal{X} \rightarrow \mathbb{R}$ with a similar interpretation to that of the ranking features, i.e., x_1 is ranked higher than x_0 by H if $H(x_1) > H(x_0)$. For the movie task, this corresponds to a complete ordering of all movies (with possible ties allowed).

The goal of the learner is to produce a "good" ranking of all instances, including those not observed in training. For instance, for the movie task, we would like to find a ranking of all movies which accurately predicts which ones a movie-viewer will like more or less than others; obviously, this ranking should include movies that the viewer has not already seen. As in other learning settings, how well the learning system performs on unseen data depends on many factors, such as the number of instances covered in training and the representational complexity of the ranking produced by the learner.

There are various methods that can be used to evaluate such a ranking. Some of these are discussed in Section 5. The boosting algorithm described in the next section attempts to minimize one possible measure called the ranking loss.

Given: initial distribution D over $\mathcal{X} \times \mathcal{X}$.

Initialize: $D_1 = D$.

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak hypothesis $h_t : \mathcal{X} \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(x_0, x_1) = \frac{D_t(x_0, x_1) \exp(\alpha_t(h_t(x_0) - h_t(x_1)))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $H(x) = \sum_{t=1}^T \alpha_t h_t(x)$.

Figure 1: The RankBoost algorithm.

3 A boosting algorithm for the ranking task

In this section, we describe an approach to the ranking problem based on a machine learning method called boosting, in particular, Freund and Schapire's [6] AdaBoost algorithm and its successor developed by Schapire and Singer [13]. Boosting is a method of producing highly accurate prediction rules by combining many "weak" rules which may be only moderately accurate.

In the current setting, we seek a learning algorithm which will produce a function $H : \mathcal{X} \rightarrow \mathbb{R}$ whose induced ordering of \mathcal{X} will approximate the relative orderings encoded by the feedback function Φ . To formalize this goal, let $D(x_0, x_1) = c \cdot \max\{0, \Phi(x_0, x_1)\}$ so that all negative entries of Φ (which carry no additional information) are set to zero. Here, c is a positive constant chosen so that

$$\sum_{x_0, x_1} D(x_0, x_1) = 1.$$

(When a specific range is not specified on a sum, we always assume summation over all of \mathcal{X} .) A pair x_0, x_1 is said to be *crucial* if $\Phi(x_0, x_1) > 0$ so that the pair receives non-zero weight under D .

Our boosting algorithm is designed to find an H with a small weighted number of crucial-pair misorderings, namely,

$$\begin{aligned} \sum_{x_0, x_1} D(x_0, x_1) \mathbb{I}[H(x_1) \leq H(x_0)] \\ = \Pr_{(x_0, x_1) \sim D} [H(x_1) \leq H(x_0)]. \end{aligned} \quad (1)$$

Here and throughout this paper, we define $\mathbb{I}[\pi]$ to be 1 if predicate π holds and 0 otherwise. We call the quantity in Eq. (1) the *ranking loss* and we denote it by $\text{rloss}_D(H)$.

3.1 The RankBoost algorithm

We call our boosting algorithm RankBoost, and its pseudocode is shown in Figure 1. Like all boosting algorithms, RankBoost operates in rounds. We assume access to a separate procedure called the *weak learner* which, on each round, is called to produce a *weak hypothesis*. RankBoost maintains a distribution D_t over $\mathcal{X} \times \mathcal{X}$ which is passed on round t to the weak learner. This distribution encodes the relative importance to the weak learner that one instance is ranked above another.

Weak hypotheses have the form $h_t : \mathcal{X} \rightarrow \mathbb{R}$. We think of these as providing ranking information in the manner described above. The weak learner we used in our experiments is based on the given ranking features; details are given in Section 4.

The boosting algorithm uses the weak hypotheses to update the distribution as shown in Figure 1. Suppose that x_0, x_1 is a crucial pair so that we want x_1 to be ranked higher than x_0 (in all other cases, D_t will be zero). Assuming for the moment that the parameter $\alpha_t > 0$ (as it usually will be), this rule has the effect of decreasing the weight $D_t(x_0, x_1)$ if h_t gives a correct ranking ($h_t(x_1) > h_t(x_0)$) and increases the weight otherwise. Thus, D_t will tend to concentrate on the pairs whose relative ranking is hardest to determine. The actual setting of α_t will be discussed shortly.

The *final* or *combined hypothesis* H is a weighted sum of the weak hypotheses. We can prove the following bound on the ranking loss of H . This theorem also provides guidance in choosing α_t and in designing the weak learner as we discuss below. Note that this theorem only concerns performance on the training data. As in more standard classification problems, the loss on a separate test set can also be theoretically bounded given appropriate assumptions using uniform-convergence theory [2, 7, 12, 15].

Theorem 1 Assuming the notation of Figure 1, the ranking loss of H is

$$\text{rloss}_D(H) \leq \prod_{t=1}^T Z_t.$$

Proof: Unraveling the update rule, we have that

$$D_{T+1}(x_0, x_1) = \frac{D(x_0, x_1) \exp(H(x_0) - H(x_1))}{\prod_t Z_t}.$$

Note that $\mathbb{I}[x \geq 0] \leq e^x$ for all real x . Therefore, the ranking loss with respect to initial distribution D is

$$\begin{aligned} \sum_{x_0, x_1} D(x_0, x_1) \mathbb{I}[H(x_0) \geq H(x_1)] \\ \leq \sum_{x_0, x_1} D(x_0, x_1) \exp(H(x_0) - H(x_1)) \\ = \sum_{x_0, x_1} D_{T+1}(x_0, x_1) \prod_t Z_t = \prod_t Z_t. \end{aligned}$$

This proves the theorem. ■

Note that RankBoost generally requires $O(|\Phi|)$ space and time per round.

3.2 Choosing α_t and criteria for weak learners

Thus to minimize ranking loss, on each round t we should choose α_t and construct weak hypotheses h_t in a manner that tends to minimize

$$Z_t = \sum_{x_0, x_1} D_t(x_0, x_1) \exp(\alpha_t(h_t(x_0) - h_t(x_1))).$$

There are various methods for achieving this end. Here we sketch three. Let us fix t and drop all t subscripts when

clear from context. (In particular, for the time being, D will denote D_t rather than an initial distribution.)

First and most generally, for any given weak hypothesis h , it can be shown that Z , viewed as a function of α , has a unique minimum which can be found numerically via a simple binary search (except in trivial degenerate cases). Details are omitted.

The second method of minimizing Z is applicable in the special case that h has range $\{0, 1\}$. In this case, we can minimize Z analytically as follows: For $b \in \{-1, 0, +1\}$, let

$$W_b = \sum_{x_0, x_1} D(x_0, x_1) [h(x_0) - h(x_1) = b].$$

Also, abbreviate W_{+1} by W_+ and W_{-1} by W_- . Then $Z = W_- e^{-\alpha} + W_0 + W_+ e^{\alpha}$. Using simple calculus, it can be verified that Z is minimized by setting $\alpha = \frac{1}{2} \ln(W_-/W_+)$ which yields $Z = W_0 + 2\sqrt{W_-W_+}$. Thus, if we are using weak hypotheses with range restricted to $\{0, 1\}$, we should attempt to find h which tends to minimize this value of Z and we should then set α accordingly.

For weak hypotheses with range $[0, 1]$, we can use a third method based on an approximation of Z . Specifically, note that

$$e^{\alpha x} \leq \left(\frac{1+x}{2}\right) e^{\alpha} + \left(\frac{1-x}{2}\right) e^{-\alpha}$$

for all real α and $x \in [-1, +1]$. Thus, we can approximate Z by

$$\begin{aligned} Z &\leq \sum_{x_0, x_1} D(x_0, x_1) \left[\left(\frac{1+h(x_0)-h(x_1)}{2}\right) e^{\alpha} \right. \\ &\quad \left. + \left(\frac{1-h(x_0)+h(x_1)}{2}\right) e^{-\alpha} \right] \\ &= \left(\frac{1-r}{2}\right) e^{\alpha} + \left(\frac{1+r}{2}\right) e^{-\alpha} \end{aligned} \quad (2)$$

where

$$r = \sum_{x_0, x_1} D(x_0, x_1) (h(x_1) - h(x_0)). \quad (3)$$

The right hand side of Eq. (2) is minimized when

$$\alpha = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right) \quad (4)$$

which, plugging into Eq. (2), yields $Z \leq \sqrt{1-r^2}$. Thus, to approximately minimize Z using weak hypotheses with range $[0, 1]$, we can attempt to maximize $|r|$ as defined in Eq. (3) and then set α as in Eq. (4). This is the method used in our experiments.

3.3 An efficient implementation for bipartite feedback

In this section, we describe a more efficient implementation of RankBoost for feedback of a special form. We say that the feedback function is *bipartite* if there exists disjoint subsets X_0 and X_1 of \mathcal{X} such that Φ ranks all instances in X_1 above all instances in X_0 and says nothing about

Given: disjoint subsets X_0 and X_1 of \mathcal{X} .

Initialize: $v_1(x) = (|X_0| |X_1|)^{-1/2}$;

$$s(x) = \begin{cases} +1 & \text{if } x \in X_1 \\ -1 & \text{if } x \in X_0 \end{cases}$$

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t (as defined by Eq. (5))
- Get weak hypothesis $h_t : \mathcal{X} \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update: $v_{t+1}(x) = \frac{v_t(x) \exp(-\alpha_t s(x) h_t(x))}{\sqrt{Z_t}}$ where $Z_t =$

$$\left(\sum_{x \in X_0} v_t(x) \exp(\alpha_t h_t(x)) \right) \left(\sum_{x \in X_1} v_t(x) \exp(-\alpha_t h_t(x)) \right).$$

Output the final hypothesis: $H(x) = \sum_{t=1}^T \alpha_t h_t(x)$.

Figure 2: A more efficient version of RankBoost for bipartite feedback.

any other pairs. That is, formally, for all $x_0 \in X_0$ and all $x_1 \in X_1$ we have that $\Phi(x_0, x_1) = +1$, $\Phi(x_1, x_0) = -1$ and Φ is zero on all other pairs.

Such feedback arises naturally, for instance, in document rank-retrieval tasks common in the field of information retrieval. Here, a set of documents may have been judged to be relevant or irrelevant, and the goal is to find a ranking of all documents which will tend to rank all relevant documents above all irrelevant documents. A feedback function which encodes these preferences will be bipartite.

If RankBoost is implemented naively as in Section 3.2, then the space and time-per-round requirements will be $\Theta(|X_0| |X_1|)$. In this section, we show how this can be improved to $O(|X_0| + |X_1|)$. Note that, in this section, $\mathcal{X}_\Phi = X_0 \cup X_1$.

The main idea is to maintain a set of weights v_t over \mathcal{X} (rather than the two-argument distribution D_t), and to maintain the condition that, on each round,

$$D_t(x_0, x_1) = v_t(x_0) v_t(x_1) \quad (5)$$

for all crucial pairs x_0, x_1 (recall that D_t is zero for all other pairs).

The pseudocode for this implementation is shown in Figure 2. Eq. (5) can be proved by induction. Details omitted for lack of space.

Finally, note that all space requirements and all per-round computations are $O(|X_0| + |X_1|)$, with the possible exception of the call to the weak learner. However, if we want the weak learner to maximize $|r|$ as in Eq. (3), then we also only need to pass $|\mathcal{X}_\Phi|$ weights to the weak learner, all of which can be computed in linear time. Omitting t subscripts, and defining $s()$ as in Figure 2, we can rewrite r as

$$r = \sum_{x_0, x_1} D(x_0, x_1) (h(x_1) - h(x_0))$$

$$\begin{aligned}
&= \sum_{x_0 \in X_0} \sum_{x_1 \in X_1} v(x_0)v(x_1) (h(x_1)s(x_1) + h(x_0)s(x_0)) \\
&= \sum_{x_0 \in X_0} \left(v(x_0) \sum_{x_1 \in X_1} v(x_1) \right) s(x_0) h(x_0) \\
&\quad + \sum_{x_1 \in X_1} \left(v(x_1) \sum_{x_0 \in X_0} v(x_0) \right) s(x_1) h(x_1) \\
&= \sum_x d(x)s(x)h(x) \tag{6}
\end{aligned}$$

where $d(x) = v(x) \sum_{x': s(x) \neq s(x')} v(x')$. All of the weights $d(x)$ can be computed in linear time by first computing the sums which appear in this equation for the two possible cases that x is in X_0 or X_1 . Thus, we only need to pass $|\mathcal{X}_\Phi|$ weights to the weak learner in this case rather than the full distribution D_t of size $|X_0| |X_1|$.

4 Weak hypotheses for ranking

As described in Section 3, our algorithm RankBoost requires access to a weak learner to produce weak hypotheses. In this section, we describe an efficient implementation of a weak learner for ranking.

Perhaps the simplest and most obvious weak learner would find a weak hypothesis h which is equal to one of the ranking features f_i , except on unranked instances. That is,

$$h(x) = \begin{cases} f_i(x) & \text{if } f_i(x) \in \mathbb{R} \\ q_{\text{def}} & \text{if } f_i(x) = \phi \end{cases}$$

for some $q_{\text{def}} \in \mathbb{R}$.

The main problem with such a weak learner is that it depends critically on the *actual values* defined by the ranking features, rather than relying exclusively on the relative-ordering information which they provide. We believe that learning algorithms of the latter form will be much more general and applicable. Such methods can be used even when features provide only an ordering of instances and no scores or other information are available. Such methods also side-step the issue of combining ranking features whose associated scores have different semantics (such as the different scores assigned to URL's by different search engines).

For these reasons, we focus in this section and in our experiments on $\{0, 1\}$ -valued weak hypotheses which use the ordering information provided by the ranking features, but ignore specific scoring information. In particular, we will use weak hypotheses h of the form

$$h(x) = \begin{cases} 1 & \text{if } f_i(x) > \theta \\ 0 & \text{if } f_i(x) \leq \theta \\ q_{\text{def}} & \text{if } f_i(x) = \phi \end{cases} \tag{7}$$

where $\theta \in \mathbb{R}$ and $q_{\text{def}} \in \{0, 1\}$. That is, a weak hypothesis is derived from a ranking feature f_i by comparing the score of f_i on a given instance to a threshold θ . To instances left unranked by f_i , the weak hypothesis assigns the default score q_{def} . For the remainder of this section, we show how to choose the “best” feature, threshold and default score.

Let us fix t and drop it from all subscripts to simplify the notation. Since the ranges of our weak hypotheses are bounded in $[0, 1]$, we can use the third method¹ described in Section 3.2 to guide us in our search for a weak hypothesis. Recall that, according to this method, the weak learner should seek a weak hypothesis which maximizes $|r|$ as given by Eq. (3). For a given candidate weak hypothesis, we can compute r directly in $O(|\Phi|)$ time. Moreover, for each of the n ranking features, there are at most $|\mathcal{X}_\Phi| + 1$ thresholds to consider (as defined by the range of f_i on \mathcal{X}_Φ) and two possible default scores (0 and 1). Thus, naively, $|r|$ can be maximized in $O(n|\Phi||\mathcal{X}_\Phi|)$ time. We now describe a time and space efficient algorithm for maximizing $|r|$ which requires only $O(n|\mathcal{X}_\Phi| + |\Phi|)$ time. (In case of bipartite feedback, if the boosting algorithm of Section 3.3 is used, only $O(n|\mathcal{X}_\Phi|)$ time is needed.)

We begin by rewriting r for a given D and h as follows:

$$\begin{aligned}
r &= \sum_{x_0, x_1} D(x_0, x_1) (h(x_1) - h(x_0)) \\
&= \sum_{x_0, x_1} D(x_0, x_1) h(x_1) - \sum_{x_0, x_1} D(x_0, x_1) h(x_0) \\
&= \sum_x h(x) \sum_{x'} D(x', x) - \sum_x h(x) \sum_{x'} D(x, x') \\
&= \sum_x h(x) \sum_{x'} (D(x', x) - D(x, x')) \\
&= \sum_x h(x) \pi(x), \tag{8}
\end{aligned}$$

where we define $\pi(x) = \sum_{x'} (D(x', x) - D(x, x'))$ as the *potential* of x . Note that $\pi(x)$ depends only on the current distribution D . Hence, the weak learner can precompute all the potentials at the beginning of each boosting round in $O(|\Phi|)$ time and $O(|\mathcal{X}_\Phi|)$ space. When the feedback is bipartite, comparing Eqs. (6) and (8), we see that $\pi(x) = d(x)s(x)$ where d and s are defined in Section 3.3; thus, in this case, π can be computed even faster in only $O(|\mathcal{X}_\Phi|)$ time.

Now let us address the problem of finding a good threshold value θ and default value q_{def} . We need to scan the candidate ranking features f_i and evaluate $|r|$ (defined by Eq. (8)) for each possible choice of f_i , θ and q_{def} . For h defined by Eq. (7), we have that

$$r = \sum_{x: f_i(x) > \theta} \pi(x) + q_{\text{def}} \sum_{x: f_i(x) = \phi} \pi(x). \tag{9}$$

For a fixed ranking feature f_i , let $\mathcal{X}_{f_i} = \{x \in \mathcal{X}_\Phi \mid f_i(x) \neq \phi\}$ be the set of feedback instances ranked by f_i . We only need to consider $|\mathcal{X}_{f_i}| + 1$ threshold values, namely, $\{f_i(x) \mid x \in \mathcal{X}_{f_i}\} \cup \{\infty\}$ since these define all possible behaviors on the feedback instances. Moreover, we can straightforwardly compute the first term of Eq. (9) for *all* thresholds in this set in time $O(|\mathcal{X}_{f_i}|)$ simply by scanning

¹Although the second method could have been used, we chose to focus on the third method because it is slightly simpler. Experiments using the second method are in our future plans.

ML Domain	Top 1	Top 2	Top 5	Top 10	Top 20	Top 30	Avg Rnk
RankBoost	102	144	173	184	194	202	4.38
Best (Top 1)	117	137	154	167	177	181	6.80
Best (Top 10)	112	147	172	179	185	187	5.33
Best (Top 30)	95	129	159	178	187	191	5.68
University Domain							
RankBoost	95	141	197	215	247	263	7.74
Best single query	112	144	198	221	238	247	8.17

Table 1: Comparison of the combined hypothesis and individual search templates.

down a presorted list of threshold values and maintaining the partial sum in the obvious way.

For each threshold, we also need to evaluate $|r|$ for the two possible assignments of q_{def} (0 or 1). To do this, we simply need to evaluate $\sum_{x: f_i(x)=\phi} \pi(x)$ once. Naively, this takes $O(|\mathcal{X}_\phi - \mathcal{X}_{f_i}|)$ time, i.e., linear in the number of *unranked* instances. We would prefer all operations to depend instead on the number of ranked instances since, in applications such as meta-searching and information retrieval, each ranking feature may rank only a small fraction of the instances. To do this, note that $\sum_x \pi(x) = 0$ by definition of $\pi(x)$. This implies that

$$\sum_{x: f_i(x)=\phi} \pi(x) = - \sum_{x: f_i(x) \neq \phi} \pi(x).$$

The right hand side of this equation can clearly be computed in $O(|\mathcal{X}_{f_i}|)$ time.

Thus, for a given ranking feature, the total time required to evaluate $|r|$ for all candidate weak hypotheses is only linear in the number of instances that are ranked by that feature.

5 Experimental evaluation of RankBoost

In this section, we report experiments with RankBoost on two ranking problems. The first is a simplified Web meta-search task, the goal of which was to build a search strategy for finding homepages of machine-learning researchers and universities. The second task is a collaborative-filtering problem of making movie recommendations for a new user based on the preferences of previous users.

In each experiment, we divided the available data into training data and test data, ran each algorithm on the training data, and evaluated the output hypothesis on the test data. Details are given below.

5.1 Meta-search task

We first present experiments on learning to combine the results of several Web searches. This problem exhibits many facets that require a general approach such as ours. For instance, approaches that learn to combine similarity scores are not applicable since the similarity scores of Web search engines are often unavailable.

In order to test RankBoost on this task, we used the data of Cohen, Schapire and Singer [4]. Their goal was to simulate the problem of building a domain-specific search engine. As test cases, they picked two fairly narrow classes

of queries—retrieving the homepages of machine-learning researchers (ML), and retrieving the homepages of universities (UNIV). They chose these test cases partly because the feedback was readily available from the Web. They obtained a list of machine-learning researchers, identified by name and affiliated institution, together with their homepages,² and a similar list for universities, identified by name and (sometimes) geographical location from Yahoo! We refer to each entry on these lists (i.e., a name-affiliation pair or a name-location pair) as a *base query*. The goal is to learn a meta-search strategy which, given a base query, will generate a ranking of URL's that includes the correct homepage at or close to the top.

Cohen, Schapire and Singer also constructed a series of special-purpose *search templates* for each domain. Each template specifies a query expansion method for converting a base query into a likely seeming AltaVista query which we call the *expanded query*. For example, one of the templates has the form `+ "NAME" +machine +learning` which means that AltaVista should search for all the words in the person's name plus the words 'machine' and 'learning'. When applied to the base query 'Joe Researcher from Learning University' this template expands to the expanded query `+ "Joe Researcher" +machine +learning`.

A total of 16 search templates were used for the ML domain and 22 for the UNIV domain. Each search template was used to retrieve the top thirty ranked documents. If none of these lists contained the correct homepage, then the base query was discarded from the experiment. In the ML domain, there were 210 base queries for which at least one search template returned the correct homepage; for the UNIV domain there were 290 such base queries.

It is instructive to see how this ranking problem can be mapped into our framework. Formally, the instances now are all pairs of the form (q, u) where q is a base query and u is one of the URL's returned by one of the search templates for this query. Each ranking feature f_i is constructed from a corresponding search template i by assigning the j th URL u on its list (for base query q) a rank of $-j$; that is, $f_i((q, u)) = -j$. If u was not ranked for this base query, then we set $f_i((q, u)) = \phi$. We also construct a separate feedback function Φ_q for each base query q which ranks the correct homepage URL u_* above all others. That is, $\Phi_q((q, u), (q, u_*)) = +1$ and $\Phi_q((q, u_*), (q, u)) = -1$ for all $u \neq u_*$. All other entries of Φ_q are set to zero. All the feedback functions Φ_q were then combined into one feedback function Φ by summing as described in Section 2.

Given this mapping of the ranking problem into our framework, we can immediately apply RankBoost. This mapping implies that each weak hypothesis is defined by a search template i (corresponding to ranking feature f_i), and a threshold value θ . Given a base query q and a URL u , this weak hypothesis outputs 1 or 0 if u is ranked above or below the threshold θ on the list of URL's returned by the expanded query associated with search template i applied to base query q . As usual, the final hypothesis H is a weighted

²From 'http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html'.

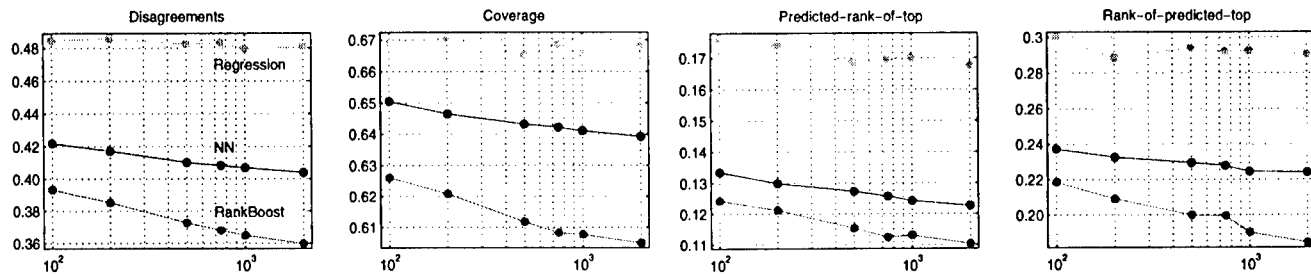


Figure 3: Performance of algorithms with respect to feature sets of sizes 100, 200, 500, 750, 1000, 2000.

sum of the weak hypotheses. Thus, given a test base query q , we first form all of the expanded queries and send these to the search engine to obtain lists of URL's. We then evaluate H as above on each pair (q, u) , where u is a returned URL, to obtain a predicted ranking of all of the URL's.

For evaluation, we divided the data into training and test sets using four-fold cross-validation. We created four partitions of the data, each one using 75% of the base queries for training and 25% for testing. Of course, the learning algorithms had no access to the test data during training.

Experimental parameters and evaluation. Since all search templates had access to the same set of documents, if a URL was not returned in the top 30 documents by a search template, we interpreted this as ranking the URL below all of the returned documents. Thus we set the parameter q_{def} , the default value for weak hypotheses, to be 0 (see Section 4).

In order to determine a good number of boosting rounds, we first ran RankBoost on each partition of the data and produced a graph of the average training error (omitted due to space constraints). On average, the training error reached zero after 85 rounds of boosting, so that is the number of boosting rounds that we used in all of the meta-search experiments.

To evaluate the performance of the individual search templates in comparison to the combined hypothesis output by RankBoost, we measured the number of queries for which the correct document was in the top k ranked documents, for various values of k . We then compared the performance of the combined hypothesis to that of the best search template for each value of k . The results for the ML and UNIV domains are shown in Table 1. All columns except the last give the number of base queries for which the correct homepage was retrieved above rank k . Bold figures give the maximum value over all of the search templates on the test data. Note that the best search template is determined based on its performance on the *test* data, while RankBoost only has access to *training* data.

For the ML data set, the combined hypothesis closely tracked the performance of the best expert at every value of k , which is especially interesting since no single template was the best for all values of k . For the UNIV data set, a single template was the best³ for all values of k , and the combined hypothesis performed almost as well as the best template for $k = 1, 2, \dots, 10$ and then outperformed the best template for $k = 20, 30$.

We also computed (an approximation to) average rank, i.e., the rank of the correct homepage URL, averaged over all base queries in the test set. Since the correct URL was sometimes not ranked or given a very high rank, we artificially assigned a rank of 31 to every document that was either unranked or ranked above rank 30. We also limited the maximum rank in the output generated by RankBoost to 31 to compensate for the fact that 31 was the maximum rank that can be assigned by any single search template.

The last column of Table 1 gives average rank. This table illustrates the robustness of the combined hypothesis on the ML domain. It outperforms the best template for all measures except top 1, where it differs from the best expert by 12%, and top 2, where it differs by 2%. On the UNIV queries, the combined hypothesis is almost always competitive with the best template for every value of k , with the exception of $k = 1$, where it trails the best expert by 15%. (Nevertheless, since this domain included such a good template, there is little reason to use something as complicated as RankBoost.)

5.2 Movie recommendations

We also tested RankBoost on the movie-recommendations task described in the introduction. For our experiments, we used publicly available⁴ data provided by the Digital Equipment Corporation which ran its own EachMovie recommendation service for eighteen months from March 1996 to September 1997 and collected user preference data. Users were able to assign a movie a score from the set $R = \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$, 1.0 being the best. We used the data of 61,625 users entering a total of 2,811,983 numeric ratings for 1,628 different movies (films and videos).

Most of the mapping of this problem into our framework was described in Section 2. For our experiments, we selected a subset C of the users to serve as ranking features: each user in C defined an ordering of the set of movies which he or she viewed. We did not set the parameter q_{def} , allowing the weak learner to choose it adaptively. The feedback function Φ was then defined as in Section 2 using the movie ratings of a single target user. We used half of the movies viewed by the selected target user for the feedback function in training, and used the other half of the viewed movies for testing as described below. We then averaged all results over many runs with many different target users. In these experiments, we ran RankBoost for 100 rounds.

³The best query expansion heuristic for the UNIV domain was "NAME" PLACE.

⁴From 'http://www.research.digital.com/SRC/eachmovie/'.

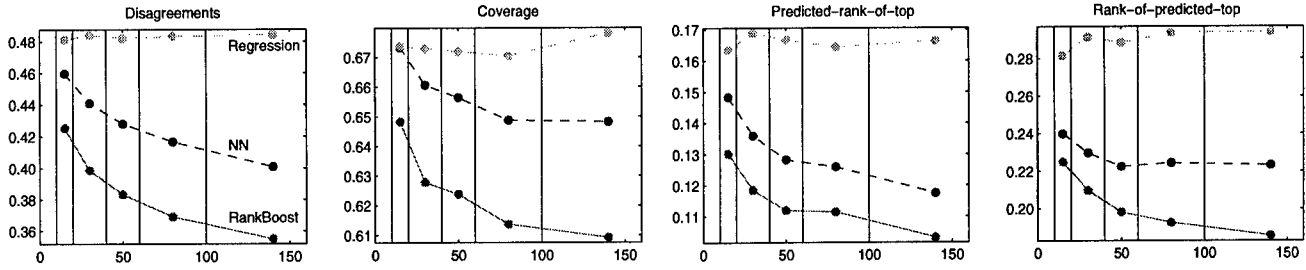


Figure 4: Performance of the algorithms on different feature densities.

We compared the performance of RankBoost on this data set to two other algorithms, a regression algorithm and a nearest-neighbor algorithm.

Regression. We used a regression algorithm similar to the ones used by Hill and others [8]. The regression algorithm employs the assumption that the preferences of a target user Alice can be described as a linear combination of the preferences of other users. Formally, let \vec{a} be a row vector whose components are the scores Alice assigned to movies (discarding unranked movies). Let C be a matrix containing the scores of the other users for the subset of movies that Alice has ranked. Since some of the users have not ranked movies that were ranked by Alice, we need to decide on a default rank for these movies. For each user represented by a row in C , we set the score of the user's unranked movies to be the user's average score over all movies. We next use linear regression to find a vector \vec{w} of minimum length which minimizes $\|\vec{w}C - \vec{a}\|$. This can be done using standard numerical techniques (we used the package available in Matlab). Given \vec{w} we can now predict the ratings of all the movies.

Nearest neighbor. Given a target user Alice with certain movie preferences, the nearest-neighbor algorithm (NN) finds a user Bob whose preferences are most similar to Alice's and then uses Bob's preferences to make recommendations for Alice. More specifically, we find the ranking feature f_i (corresponding to one of the other movie viewers) which gives an ordering most similar to that of the target user as encoded by the feedback function Φ . The measure of similarity we use is the ranking loss of f_i with respect to the same initial distribution D which was constructed by RankBoost. Thus, in some sense, NN can be viewed as a single weak hypothesis output after one round of RankBoost (although no threshold of f_i is performed).

A problem with this algorithm is that the user it selects may not rank all the movies ordered by the target user. To fix this, we modified NN to associate with each feature f_i a default rank $q_{\text{def}} \in R$ which f_i assigns to unranked movies. When searching for the best feature, NN chooses q_{def} by calculating and then minimizing the ranking loss for each possible value of q_{def} . If it is the case that this user ranks all of the movies seen by the target user, then NN sets q_{def} to the average rank over all movies that it ranked (including those not ranked by the target user).

In order to evaluate and compare performance, we used four different error measures. We assume that the learning system produces a real-valued function H which orders instances in the usual way (x_1 ranked higher than x_0 if

$H(x_1) > H(x_0)$). We compare the ordering of H to a "correct" ordering c over test instances, also represented formally as a real-valued function. For simplicity, we here only give definitions for these measures when H defines a *total* order of all instances so that no ties occur in either order. The definitions can be extended by assuming that ties are broken randomly and taking expectations (details omitted for lack of space).

All our measures have range $[0, 1]$, with a value 0 being a "perfect" score.

Disagreement. Disagreement is the fraction of distinct pairs of instances which are misordered by H (with respect to c). If c were used to construct a feedback function, this would be equivalent to the ranking loss of H .

Predicted-rank-of-top (PROT). This is the minimum rank (according to H) of any of the truly top-rated instances (according to c). The score is then rescaled to have a possible range of $[0, 1]$.

Coverage. This is the *maximum* rank (according to H) of any of the truly top-rated instances (according to c). The score is then rescaled to have a possible range of $[0, 1]$. (Note that coverage and PROT are equal if there is a unique top-rated instances according to c .)

Rank-of-predicted-top (ROPT). This is the number of instances ranked strictly higher (according to c) than the predicted top-rated instance (according to H). The score is then rescaled to have a possible range of $[0, 1]$.

We now describe our experimental results. We ran a series of three tests, examining the performance of the algorithms as we varied the number of features, the *density* of the features (number of movies ranked by each user), and the density of the feedback.

We first experimented with the number of features used for ranking. We selected two disjoint random sets T and T' of 2000 users each. We further divided T into six subsets T_1, T_2, \dots, T_6 of respective sizes 100, 200, 500, 750, 1000, 2000, such that $T_1 \subset T_2 \subset \dots \subset T_6$. Each T_j served as a feature set for training on half of a target user's movies and testing on the other half, for each user in T' . For each algorithm, we calculated the measures described above averaged over the 2000 test users. We ran the algorithms on five disjoint random splits of the data into feature and feedback sets, and we averaged the results, which are shown in Figure 3.

RankBoost was the clear winner for all four performance measures. The performance of regression was much poorer, and NN was in between. For the most part, the performance of the algorithms improved as the number of

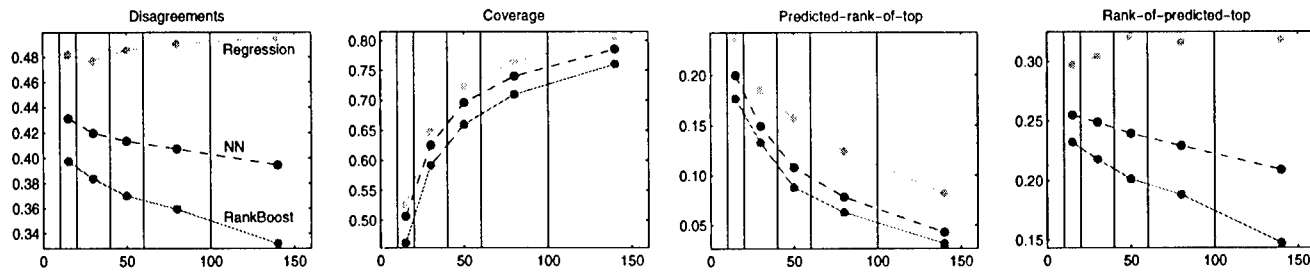


Figure 5: Performance of algorithms on different feedback densities.

features increased. RankBoost and NN did reasonably well with respect to disagreement, which they directly tried to minimize, while regression's error rate was just slightly better than 50%. All three algorithms did well on PROT and ROPT, although again regression was worse (about 30% worse than RankBoost). All three algorithms had difficulty with coverage. In all cases, RankBoost was better able to use the increased number of features.

We next explored the effect of the features and feedback density, the number of movies ranked by each user. We partitioned the set of features into bins according to their density. The bins were 10-20, 21-40, 41-60, 61-100, 101-1455, where 1455 was the maximum number of movies ranked by a single user in the data set. We selected a random set of 1000 features (users) from each bin to be evaluated on a disjoint random set of 1000 feedback users (of varying densities). We ran the algorithms on six such random splits, calculated the averages of the four error measures on each split, and then averaged them together. The results are shown in Figure 4. The x -coordinate of each point is the average density of the features in a single bin; for example, 80 is the average density of features whose density is in the range 61-100. The relative performance of the algorithms was the same as in Figure 3. RankBoost was again able to use the denser features to obtain lower error rates, while the improvement of NN was less dramatic. Regression actually performed the same or worse as the feature density increased.

We varied the feedback densities in the same way as the feature densities. We used a random set of 1000 features and again ran on six random splits, taking averages. The results appear in Figure 5. As feedback density increased, RankBoost and NN improved with respect to disagreement and ROPT, while regression performed worse. All three algorithms did well on PROT, as might be expected, since larger feedback sets will likely have many top-ranked movies. For the same reason, all three algorithms were very poor on coverage.

We see from these graphs that RankBoost performed the best on this ranking task. RankBoost's approach of ordering based on *relative* comparisons performed much better than regression which treats the movie scores as absolute numerical values. RankBoost also improved on the nearest-neighbor algorithm by combining *multiple* features to form an accurate prediction rule.

References

- [1] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew.

- Automatic combination of multiple ranked retrieval systems. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.
- [2] Peter L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 1998 (to appear).
- [3] Rich Caruana, Shumeet Baluja, and Tom Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *Advances in Neural Information Processing Systems 8*, pages 959-965, 1996.
- [4] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. In *Advances in Neural Information Processing Systems 10*, 1998.
- [5] O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madani, and O. Waarts. Efficient information gathering on the internet. In *37th Annual Symposium on Foundations of Computer Science*, 1996.
- [6] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119-139, August 1997.
- [7] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78-150, 1992.
- [8] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Human Factors in Computing Systems CHI'95 Conference Proceedings*, pages 194-201, 1995.
- [9] Paul Resnick, Neophytos Iacovou, Mitesh Sushak, Peter Bergstrom, and John Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of Computer Supported Cooperative Work*, 1995.
- [10] Gerard Salton. *Automatic text processing: the transformation, analysis and retrieval of information by computer*. Addison-Wesley, 1989.
- [11] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
- [12] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, to appear.
- [13] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 1998.
- [14] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Human Factors in Computing Systems CHI'95 Conference Proceedings*, 1995.
- [15] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

Bayesian Network Classification with Continuous Attributes: Getting the Best of Both Discretization and Parametric Fitting

Nir Friedman*

Computer Science Division
University of California
Berkeley, CA 94920
nir@cs.berkeley.ed

Moises Goldszmidt

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
moises@sri.com

Thomas J. Lee

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
tomlee@sri.com

Abstract

In a recent paper, Friedman, Geiger, and Goldszmidt [8] introduced a classifier based on Bayesian networks, called *Tree Augmented Naive Bayes* (TAN), that outperforms naive Bayes and performs competitively with C4.5 and other state-of-the-art methods. This classifier has several advantages including robustness and polynomial computational complexity. One limitation of the TAN classifier is that it applies only to discrete attributes, and thus, continuous attributes must be discretized. In this paper, we extend TAN to deal with continuous attributes directly via parametric (e.g., Gaussians) and semiparametric (e.g., mixture of Gaussians) conditional probabilities. The result is a classifier that can represent and combine both discrete and continuous attributes. In addition, we propose a new method that takes advantage of the modeling language of Bayesian networks in order to represent attributes both in discrete and continuous form simultaneously, and use both versions in the classification. This automates the process of deciding which form of the attribute is most relevant to the classification task. It also avoids the commitment to either a discretized or a (semi)parametric form, since different attributes may correlate better with one version or the other. Our empirical results show that this latter method usually achieves classification performance that is as good as or better than either the purely discrete or the purely continuous TAN models.

1 INTRODUCTION

The effective handling of continuous attributes is a central problem in machine learning and pattern recognition. Almost every real-world domain, including medicine, industrial control, and finance, involves continuous attributes. Moreover, these attributes usually have rich interdependencies with other discrete attributes. Many approaches in machine learning deal with continuous attributes by discretizing them. In statistics and pattern recognition, on the other hand, the typical approach is to use a parametric family of distributions (e.g. Gaussians) to model the data.

Each of these strategies has its advantages and disadvantages. By using a specific parametric family, we are making strong assumptions about the nature of the data. If these assumptions are warranted, then the induced model can be a

good approximation of the data. In contrast, discretization procedures are not bound by a specific parametric distribution; yet they suffer from the obvious loss of information. Of course, one might argue that for specific tasks, such as classification, it suffices to estimate the probability that the data falls in a certain range, in which case discretization is an appropriate strategy.

In this paper, we introduce an innovative approach for dealing with continuous attributes that avoids a commitment to either one of the strategies outlined above. This approach uses a dual representation for each continuous attribute: one discretized, and the other based on fitting a parametric distribution. We use Bayesian networks to model the interaction between the discrete and continuous versions of the attribute. Then, we let the learning procedure decide which type of representation best models the training data and what interdependencies between attributes are appropriate. Thus, if attribute B can be modeled as a linear Gaussian depending on A , then the network would have a direct edge from A to B . On the other hand, if the parametric family cannot fit the dependency of B on A , then the network might use the discretized representation of A and B to model this relation. Note that the resulting models can (and usually do) involve both parametric and discretized models of interactions among attributes.

In this paper we focus our attention on classification tasks. We extend a Bayesian network classifier, introduced by Friedman, Geiger, and Goldszmidt (FGG) [8] called "Tree Augmented Naive Bayes" (TAN). FGG show that TAN outperforms naive Bayes, yet at the same time maintains the computational simplicity (no search involved) and robustness that characterize naive Bayes. They tested TAN on problems from the UCI repository [16], and compared it to C4.5, naive Bayes, and wrapper methods for feature selection with good results. The original version of TAN is restricted to multinomial distributions and discrete attributes. We start by extending the set of distributions that can be represented in TAN to include Gaussians, mixtures of Gaussians, and linear models. This extension results in classifiers that can deal with a combination of discrete and continuous attributes and model interactions between them. We compare these classifiers to the original TAN on several UCI data sets. The results show that neither approach dominates the other in terms of classification accuracy.

We then augment TAN with the capability of representing

*Current address: Institute of Computer Science, The Hebrew University, Givat Ram, Jerusalem 91904, Israel, nir@cs.huji.ac.il.

each continuous attribute in both parametric and discretized forms. We examine the consequences of the dual representation of such attributes, and characterize conditions under which the resulting classifier is well defined. Our main hypothesis is that the resulting classifier will usually achieve classification performance that is as good or better than both the purely discrete and purely continuous TAN models. This hypothesis is supported by our experiments.

We note that this dual representation capability also has ramifications in tasks such as density estimation, clustering, and compression, which we are currently investigating and some of which we discuss below. The extension of the dual representation to arbitrary Bayesian networks, and the extension of the discretization approach introduced by Friedman and Goldszmidt [9] to take the dual representation into account, are the subjects of current research.

2 REVIEW OF TAN

In this discussion we use capital letters such as X, Y, Z for variable names, and lower-case letters such as x, y, z to denote specific values taken by those variables. Sets of variables are denoted by boldface capital letters such as $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, and assignments of values to the variables in these sets are denoted by boldface lowercase letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$.

A *Bayesian network* over a set of variables $\mathbf{X} = \{X_1, \dots, X_n\}$ is an annotated directed acyclic graph that encodes a joint probability distribution over \mathbf{X} . Formally, a Bayesian network is a pair $B = \langle G, \mathcal{L} \rangle$. The first component, G , is a directed acyclic graph whose vertices correspond to the random variables X_1, \dots, X_n , and whose edges represent direct dependencies between the variables. The second component of the pair, namely \mathcal{L} , represents a set of local *conditional probability distributions* (CPDs) L_1, \dots, L_n , where the CPD for X_i maps possible values x_i of X_i and $\text{pa}(X_i)$ of $\text{Pa}(X_i)$, the set of parents of X_i in G , to the conditional probability (density) of x_i given $\text{pa}(X_i)$. A Bayesian network B defines a unique joint probability distribution (density) over \mathbf{X} given by the product

$$P_B(X_1, \dots, X_n) = \prod_{i=1}^n L_i(X_i | \text{Pa}(X_i)). \quad (1)$$

When the variables in \mathbf{X} take values from finite discrete sets, we typically represent CPDs as tables that contain parameters $\theta_{x_i | \text{pa}(X_i)}$ for all possible values of X_i and $\text{Pa}(X_i)$. When the variables are continuous, we can use various parametric and semiparametric representations for these CPDs.

As an example, let $\mathbf{X} = \{A_1, \dots, A_n, C\}$, where the variables A_1, \dots, A_n are the *attributes* and C is the *class* variable. Consider a graph structure where the class variable is the root, that is, $\text{Pa}(C) = \emptyset$, and each attribute has the class variable as its unique parent, namely, $\text{Pa}(A_i) = \{C\}$ for all $1 \leq i \leq n$. For this type of graph structure, Equation 1 yields $\Pr(A_1, \dots, A_n, C) = \Pr(C) \cdot \prod_{i=1}^n \Pr(A_i | C)$. From the definition of conditional probability, we get $\Pr(C | A_1, \dots, A_n) = \alpha \cdot \Pr(C) \cdot \prod_{i=1}^n \Pr(A_i | C)$, where α is a normalization constant. This is the definition of the *naive Bayesian classifier* commonly found in the literature [5].

The naive Bayesian classifier has been used extensively for classification. It has the attractive properties of being robust and easy to learn—we only need to estimate the CPDs $\Pr(C)$ and $\Pr(A_i | C)$ for all attributes. Nonetheless, the naive Bayesian classifier embodies the strong independence

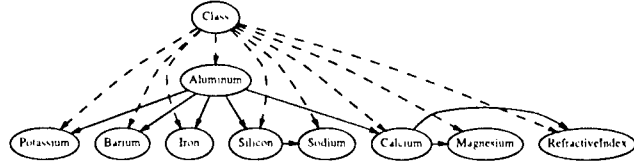


Figure 1: A TAN model learned for the data set "glass2." The dashed lines represent edges required by the naive Bayesian classifier. The solid lines are the tree augmenting edges representing correlations between attributes.

assumption that, given the value of the class, attributes are independent of each other. FGG [8] suggest the removal of these independence assumptions by considering a richer class of networks. They define the TAN Bayesian classifier that learns a network in which each attribute has the class and at most one other attribute as parents. Thus, the dependence among attributes in a TAN network will be represented via a tree structure. Figure 1 shows an example of a TAN network.

In a TAN network, an edge from A_i to A_j implies that the influence of A_i on the assessment of the class also depends on the value of A_j . For example, in Figure 1, the influence of the attribute "Iron" on the class C depends on the value of "Aluminum," while in the naive Bayesian classifier the influence of each attribute on the class is independent of other attributes. These edges affect the classification process in that a value of "Iron" that is typically surprising (i.e., $P(i|c)$ is low) may be unsurprising if the value of its correlated attribute, "Aluminum," is also unlikely (i.e., $P(i|c, a)$ is high). In this situation, the naive Bayesian classifier will overpenalize the probability of the class by considering two unlikely observations, while the TAN network of Figure 1 will not do so, and thus will achieve better accuracy.

TAN networks have the attractive property of being learnable in polynomial time. FGG pose the learning problem as a search for the TAN network that has the highest *likelihood* $LL(B : D) = P_B(D)$, given the data D . Roughly speaking, networks with higher likelihood match the data better. FGG describe a procedure *Construct-TAN* for learning TAN models and show the following theorem.

Theorem 2.1: [8] *Let D be a collection of N instances of C, A_1, \dots, A_n . The procedure Construct-TAN builds a TAN network B that maximizes $LL(B : D)$ and has time complexity $O(n^2 \cdot N)$.*

The TAN classifier is related to the classifier introduced by Chow and Liu [2]. That method learns a different tree for each class value. FGG's results show that the TAN and Chow and Liu's classifier perform roughly the same. In domains where there is substantial differences in the interactions between attributes for different class values, Chow and Liu's method performs better. In others, it is possible to learn a better tree by pooling the examples from different classes as done by TAN. Although we focus on extending the TAN classifier here, all of our ideas easily apply to classifiers that learn a different tree for each class value.

3 GAUSSIAN TAN

The TAN classifier, as described by FGG, applies only to discrete attributes. In experiments run on data sets with

continuous attributes, FGG use the prediscrretization described by Fayyad and Irani [7] before learning a classifier. In this paper, we attempt to model the continuous attributes directly within the TAN network. To do so, we need to learn CPDs for continuous attributes. In this section, we discuss Gaussian distributions for such CPDs. The theory of training such representations is standard (see, for example, [1, 5]). We only review the indispensable concepts.

A more interesting issue pertains to the structure of the network. As we shall see, when we mix discrete and continuous attributes, the algorithms must induce directed trees. This is in contrast to the procedure of FGG, which learns undirected trees and then arbitrarily chooses a root to define edge directions. We describe the procedure for inducing directed trees next.

3.1 THE BASIC PROCEDURE

We now extend the TAN algorithm for directed trees. This extension is fairly straight forward and similar ideas have been suggested for learning tree-like Bayesian networks [12]. For completeness, and to facilitate later extensions, we rederive the procedure from basic principles. Assume that we are given a data set D that consists of N identically and independently distributed (i.i.d.) instances that assign values to A_1, \dots, A_n and C . Also assume that we have specified the class of CPDs that we are willing to consider. The objective is, as before, to build a network that maximizes the likelihood function $LL(B : D) = \log P_B(D)$.

Using Eq. (1) and the independence of training instances, it is easy to show that

$$\begin{aligned} LL(B : D) &= \sum_i \sum_{j=1}^N \log L_i(x_i^j | \mathbf{Pa}(X_i)^j) \\ &= \sum_i S(X_i | \mathbf{Pa}(X_i) : L_i), \end{aligned} \quad (2)$$

where x_i^j and $\mathbf{Pa}(X_i)^j$ are the values of X_i and $\mathbf{Pa}(X_i)$ in the j th instance in D . We denote by $S(X_i | \mathbf{Pa}(X_i))$ the value attained by $S(X_i | \mathbf{Pa}(X_i), L_i)$ when L_i is the optimal CPD for this family, given the data, and the set of CPDs we are willing to consider (e.g., all tables, or all Gaussian distributions). "Optimal" should be understood in terms of maximizing the likelihood function in Eq. (2).

We now recast this decomposition in the special class of TAN networks. Recall that in order to induce a TAN network, we need to choose for each attribute A_i at most one parent other than the class C . We represent this selection by a function $\pi(i)$, s.t., if $\pi(i) = 0$, then C is the only parent of A_i , otherwise both $A_{\pi(i)}$ and C are the parents of A_i . We define $LL(\pi : D)$ to be the likelihood of the TAN model specified by π , where we select an optimal CPD for each parent set specified by π . Rewriting Eq. (2), we get

$$\begin{aligned} LL(\pi : D) &= \sum_{i, \pi(i) > 0} S(A_i | C, A_{\pi(i)}) + \\ &\quad \sum_{i, \pi(i) = 0} S(A_i | C) + S(C | \emptyset) \\ &= \sum_{i, \pi(i) > 0} (S(A_i | C, A_{\pi(i)}) - S(A_i | C)) + \\ &\quad \sum_i S(A_i | C) + S(C | \emptyset) \\ &= \sum_{i, \pi(i) > 0} (S(A_i | C, A_{\pi(i)}) - S(A_i | C)) + c, \end{aligned}$$

where c is some constant that does not depend on π . Thus, we need to maximize only the first term. This maximization

can be reduced to a graph-theoretic maximization by the following procedure, which we call Directed-TAN:

1. Initialize an empty graph \mathcal{G} with n vertices labeled $1, \dots, n$.
2. For each attribute A_i , find the best scoring CPD for $P(A_i | C)$ and compute $S(A_i | C)$. For each A_j with $j \neq i$, if an arc from A_j to A_i is legal, then find the best CPD for $P(A_i | C, A_j)$, compute $S(A_i | C, A_j)$, and add to \mathcal{G} an arc $j \rightarrow i$ with weight $S(A_i | C, A_j) - S(A_i | C)$.
3. Find a set of arcs \mathcal{A} that is a *maximal weighted branching* in \mathcal{G} . A branching is a set of edges that have at most one member pointing into each vertex and does not contain cycles. Finding a maximally weighed branching is a standard graph-theoretic problem that can be solved in low-order polynomial time [6, 17].
4. Construct the TAN model that contains arc from C to each A_i , and arc from A_j to A_i if $j \rightarrow i$ is in \mathcal{A} . For each A_i , assign it the best CPD found in step 2 that matches the choice of arcs in the branching.

From the arguments we discussed above it is easy to see that this procedure constructs the TAN model with the highest score. We note that since we are considering directed edges, the resulting TAN model might be a forest of directed trees instead of a spanning tree.

Theorem 3.1: *The procedure Directed-TAN constructs a TAN network B that maximizes $LL(B : D)$ given the constraints on the CPDs in polynomial time.*

In the next sections we describe how to compute the optimal S for different choices of CPDs that apply to different types of attributes.

3.2 DISCRETE ATTRIBUTES

Recall that if A_i is discrete, then we model $P(A_i | \mathbf{Pa}(A_i))$ by using tables that contain a parameter $\theta_{a_i | \mathbf{pa}(A_i)}$ for each choice of values for A_i and its parents. Thus,

$$\begin{aligned} S(A_i | \mathbf{Pa}(A_i)) &= \sum_j \log P(a_i^j | \mathbf{pa}(A_i)^j) \\ &= N \sum_{a_i, \mathbf{pa}(A_i)} \hat{P}(a_i, \mathbf{pa}(A_i)) \log \theta_{a_i | \mathbf{pa}(A_i)}, \end{aligned}$$

where $\hat{P}(\cdot)$ is the empirical frequency of events in the training data. Standard arguments show that the maximum likelihood choice of parameters is $P(x | y) = \hat{P}(x | y)$. Making the appropriate substitution above, we get a nice information-theoretic interpretation of the weight of the arc from A_i to A_j , $S(A_i | C, A_j) - S(A_i | C) = N \cdot I(A_i; A_j | C)$. The $I()$ term is the *conditional mutual information* between A_i and A_j given C [3]. Roughly speaking, it measures how much information A_j provides about A_i if already know the value of C . In this case, our procedure reduces to Construct-TAN of FGG, except that they use $I(A_i; A_j | C)$ directly as the weight on the arcs, while we multiply these weights by N .

3.3 CONTINUOUS ATTRIBUTES

We now consider the case where X is continuous. There are many possible parametric models for continuous variables. Perhaps the easiest one to use is the Gaussian

distribution. A continuous variable is a Gaussian with mean μ and variance σ^2 if the pdf of X has the form $\varphi(x : \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. If all the parents of a continuous A_i are discrete, then we learn a conditional Gaussian CPD [11, 15] by assigning to A_i different mean $\mu_{a_i|\mathbf{pa}(A_i)}$ and variance $\sigma^2_{a_i|\mathbf{pa}(A_i)}$ for each joint value of its parents. Standard arguments (e.g., see [1]) show that we can rewrite $S(A_i | \mathbf{Pa}(A_i))$ as a function of $E[A_i | \mathbf{pa}(A_i)]$ and $E[A_i^2 | \mathbf{pa}(A_i)]$ —the expectations of A_i and A_i^2 in these instances of the data where $\mathbf{Pa}(A_i)$ take a particular value. Standard arguments also show that we maximize the likelihood score of the CPD by choosing

$$\begin{aligned}\mu_{a_i|\mathbf{pa}(A_i)} &= E[A_i | \mathbf{pa}(A_i)] \\ \sigma^2_{a_i|\mathbf{pa}(A_i)} &= E[A_i^2 | \mathbf{pa}(A_i)] - E^2[A_i | \mathbf{pa}(A_i)].\end{aligned}$$

When we learn TAN models in domains with many continuous attributes, we also want to have families where one continuous attribute is a parent of another continuous attribute. In the Gaussian model, we can represent such CPDs by using a linear Gaussian relation. In this case, the mean of A_i depends, in a linear fashion, on the value of A_j . This relationship is parameterized by three parameters: $\alpha_{A_i|A_j,c}$, $\beta_{A_i|A_j,c}$ and $\sigma^2_{A_i|A_j,c}$ for each value c of the class variable. The conditional probability for this CPD is a Gaussian with mean $\alpha_{A_i|A_j,c} + A_j\beta_{A_i|A_j,c}$ and variance $\sigma^2_{A_i|A_j,c}$. Again, by using standard arguments, it is easy to show that $S(A_i | A_j, C)$ is a function of low-order statistics in the data, and that the maximal likelihood parameters are

$$\begin{aligned}\beta_{A_i|A_j,c} &= \frac{E[A_i A_j | c] - E[A_i | c]E[A_j | c]}{E[A_j^2 | c] - E^2[A_j | c]} \\ \alpha_{A_i|A_j,c} &= E[A_i | c] - \beta_{A_i|A_j,c} * E[A_j | c] \\ \sigma^2_{A_i|A_j,c} &= E[A_i^2 | c] - E^2[A_i | c] - \frac{(E[A_i A_j | c] - E[A_i | c]E[A_j | c])^2}{E[A_j^2 | c] - E^2[A_j | c]}\end{aligned}$$

In summary, to estimate parameters and to evaluate the likelihood, we need only to collect the statistics of each pair of attributes with the class, that is, terms of the form $E[A_i | a_j, c]$ and $E[A_i A_j | c]$. Thus, learning in the case of continuous Gaussian attributes can be done efficiently in a single pass over the data.

When we learn TAN models that contain discrete and Gaussian attributes, we restrict ourselves to arcs between discrete attributes, arcs between continuous attributes, and arcs from discrete attributes to continuous ones. If we want also to model arcs from continuous to discrete, then we need to introduce additional types of parametric models, such as logistic regression [1]. As we will show, an alternative solution is provided by the dual representation approach introduced in this paper.

3.4 SMOOTHING

One of the main risks in parameter estimation is overfitting. This can happen when the parameter in question is learned from a very small sample (e.g., predicting A_i from values of A_j and of C that are rare in the data). A standard approach to this problem is to *smooth* the estimated parameters. Smoothing ensures that the estimated parameters will

not be overly sensitive to minor changes in the training data. FGJ show that in the case of discrete attributes, smoothing can lead to dramatic improvement in the performance of the TAN classifier. They use the following smoothing rule for the discrete case

$$\theta_{a_i|\mathbf{pa}(A_i)} = \frac{N \cdot \hat{P}(\mathbf{pa}(A_i)) \hat{P}(a_i|\mathbf{pa}(A_i)) + s \cdot \hat{P}(a_i)}{N \cdot \hat{P}(\mathbf{pa}(A_i)) + s}$$

where s is a parameter that controls the magnitude of the smoothing (FGJ use $s = 5$ in all of their experiments.) This estimate uses a linear combination of the maximum likelihood parameters and the unconditional frequency of the attribute. It is easy to see that this prediction biases the learned parameters in a manner that depends on the weight of the smoothing parameter and the number of “relevant” instances in the data. This smoothing operation is similar to (and motivated by) well-known methods in statistics such as *hierarchical Bayesian* and *shrinkage* methods [10].

We can think of this smoothing operation as pretending that there are s additional instances in which A_j is distributed according to its marginal distribution. This immediately suggests how to smooth in the Gaussian case: we pretend that for these additional s samples A_i , A_i^2 have the same average as what we encounter in the totality of the training data. Thus, the statistics from the augmented data are

$$\begin{aligned}\hat{E}[A_i | \mathbf{pa}(A_i)] &= \frac{N \cdot \hat{P}(\mathbf{pa}(A_i)) E[A_i | \mathbf{pa}(A_i)] + s \cdot E[A_i]}{N \cdot \hat{P}(\mathbf{pa}(A_i)) + s} \\ \hat{E}[A_i^2 | \mathbf{pa}(A_i)] &= \frac{N \cdot \hat{P}(\mathbf{pa}(A_i)) E[A_i^2 | \mathbf{pa}(A_i)] + s \cdot E[A_i^2]}{N \cdot \hat{P}(\mathbf{pa}(A_i)) + s}\end{aligned}$$

We then use these adjusted statistics for estimating the mean and variance of A_i given its parents. The same basic smoothing method applies for estimating linear interactions between continuous attributes.

4 SEMIPARAMETRIC ESTIMATION

Parametric estimation methods assume that the data is (approximately) distributed according to a member of the given parametric family. If the data behaves differently enough, then the resulting classifier will degrade in performance. For example, suppose that for a certain class c , the attribute A_i has bimodal distribution, where the two modes x_1 and x_2 are fairly far apart. If we use a Gaussian to estimate the distribution of A_i given C , then the mean of the Gaussian would be in the vicinity of $\mu = \frac{x_1 + x_2}{2}$. Thus, instances where A_i has a value near μ would receive a high probability, given the class c . On the other hand, instances where A_i has a value in the vicinity of either x_1 or x_2 would receive a much lower probability given c . Consequently, the support c gets from A_i behaves exactly the opposite of the way it should. It is not surprising that in our experimental results, Gaussian TAN occasionally performed much worse than the discretized version (see Table 1).

A standard way of dealing with such situations is to allow the classifier more flexibility in the type of distributions it learns. One approach, called semiparametric estimation, learns a collection of parametric models. In this approach, we model $P(A_i | \mathbf{Pa}(A_i))$ using a mixture of Gaussian distributions: $P(A_i | \mathbf{pa}(A_i)) = \sum_j \varphi(A_i :$

$\mu_{A_i|\mathbf{pa}(A_i),j}, \sigma^2_{A_i|\mathbf{pa}(A_i),j})w_{A_i|\mathbf{pa}(A_i),j}$, where the parameters specify the mean and variance of each Gaussian in the mixture and $w_{A_i|\mathbf{pa}(A_i),j}$ are the weights of the mixture components. We require that the $w_{A_i|\mathbf{pa}(A_i),j}$ sum up to 1, for each value of $\mathbf{Pa}(A_i)$.

To estimate $P(A_i | \mathbf{pa}(A_i))$, we need to decide on the number of mixture components (the parameter j in the equation above) and on the best choice of parameters for that mixture. This is usually done in two steps. First, we attempt to fit the best parameters for different number of components (e.g., $j = 1, 2, \dots$), and then select an instantiation for j based on a performance criterion.

Because there is no closed form for learning the parameters we need to run a search procedure such as the Expectation-Maximization (EM) algorithm. Moreover, since EM usually finds local maxima, we have to run it several times, from different initial points, to ensure that we find a good approximation to the best parameters. This operation is more expensive than parametric fitting, since the training data cannot be summarized for training the mixture parameters. Thus, we need to perform many passes over the training data to learn the parameters. Because of space restrictions we do not review the EM procedure here, and refer the reader to [1, pp. 65–73].

With regard to selecting the number of components in the mixture, it is easy to see that a mixture with $k+1$ components can easily attain the same or better likelihood as any mixture with k components. Thus, the likelihood (of the data) alone is not a good performance criterion for selecting mixture components, since it always favors models with a higher number of components, which results in overfitting. Hence, we need to apply some form of model selection. The two main approaches to model selection are based on cross-validation to get an estimate of true performance for each choice of k , or on penalizing the performance on the training data to account for the complexity of the learned model. For simplicity, we use the latter approach with the BIC/MDL penalization. This rule penalizes the score of each mixture with $\frac{\log N}{2} 3k$, where k is the number of mixture components, and N is the number of training examples for this mixture (i.e., the number of instances in the data with this specific value of the discrete parents).

Once more, smoothing is crucial for avoiding overfitting. Because of space considerations we will not go into the details. Roughly speaking, we apply the Gaussian smoothing operation described above in each iteration of the EM procedure. Thus, we assume that each component in the mixture has a preassigned set of s samples to fit.

As our experimental results show, the additional flexibility of the mixture results in drastically improved performance in the cases where the Gaussian TAN did poorly (see, for example, the accuracy of the data sets “anneal-U” and “balance-scale” in Table 1). In this paper, we learned mixtures only when modeling a continuous feature with discrete parents. We note, however, that learning a mixture of linear models is a relatively straightforward extension that we are currently implementing and testing.

5 DUAL REPRESENTATION

The classifiers we have presented thus far require us to make a choice. We can either discretize the attributes and use the discretized TAN, or we can learn a (semi)parametric density model for the continuous attributes. Each of these methods has its advantages and problems: Discretization works well with nonstandard densities, but clearly loses much information about the features. Semiparametric estimation can work well for “well-behaved” multimodal densities. On the other hand, although we can approximate any distribution with a mixture of Gaussians, if the density is complex, then we need a large number of training instances to learn a mixture with large number of components, with sufficient confidence.

The choice we are facing is not a simple binary one, that is, to discretize or not to discretize all the attributes. We can easily imagine situations in which some of several attributes are better modeled by a semiparametric model, and others are better modeled by a discretization. Thus, we can choose to discretize only a subset of the attributes. Of course, the decision about one attribute is not independent of how we represent other attributes. This discussion suggests that we need to select a subset of variables to discretize, that is, to choose from an exponential space of options.

In this section, we present a new method, called *hybrid TAN*, that avoids this problem by representing both the continuous attributes and their discretized counterparts within the same TAN model. The structure of the TAN model determines whether the interaction between two attributes is best represented via their discretized representation, their continuous representation, or a hybrid of the discrete representation of one and the continuous representation of the other. Our hypothesis is that hybrid TAN allows us to achieve performance that is as good as either alternative. Moreover, the cost of learning hybrid TAN is about the same as that of learning either alternative.

Let us assume, that the first k attributes, A_1, \dots, A_k , are the continuous attributes in our domain. We denote by A_1^*, \dots, A_k^* the corresponding discretized attributes (i.e., A_1^* is the discretized version of A_1), based on a predetermined discretization policy (e.g., using a standard method, such as Fayyad and Irani’s [7]). Given this semantics for the discretized variables, we know that that each A_i^* is a *deterministic* function of A_i . That is, A_i^* state corresponds to the interval $[x_1, x_2]$ if and only if $A_i \in [x_1, x_2]$. Thus, even though the discretized variables are not observed in the training data, we can easily augment the training data with the discretized version of each continuous attribute.

At this stage one may consider the application of one of the methods we described above to the augmented training set. This, however, runs the risk of “double counting” the evidence for classification provided by the duplicated attributes. The likelihood of the learned model will contain a penalty for both the continuous and the discrete versions of the attribute. Consequently, during classification, a “surprising” value of an attribute would have twice the (negative) effect on the probability of the class variable. One could avoid this problem by evaluating only the likelihood assigned to the continuous version of the attributes. Unfortunately, in this case the basic decomposition of Eq. (2) no

longer holds, and we cannot use the TAN procedure.

5.1 MODELING THE DUAL REPRESENTATION

Our approach takes advantage of Bayesian networks to model the interaction between an attribute and its discretized version. We constrain the networks we learn to match our model of the discretization, that is, a discretized attribute is a function of the continuous one. More specifically, for each continuous attribute A_i , we require that $P_B(A_i^* | A_i) = 1$ iff A_i is in the range specified by A_i^* . It is easy to show (using the chain rule) that this constraint implies that $P_B(A_1, \dots, A_n, A_1^*, \dots, A_k^*) = P_B(A_1, \dots, A_n)$ avoiding the problem outlined in the previous paragraph.

Note that by imposing this constraint we are not requiring in any way that A_i be a parent of A_i^* . However, we do need to ensure that $P(A_i^* | A_i)$ is deterministic in the learned model. We do so by requiring that A_i and A_i^* are adjacent in the graph (i.e., one is the parent of the other) and by putting restrictions on the models we learn for $P(A_i | A_i^*)$ and $P(A_i^* | A_i)$. There are two possibilities:

If $A_i \rightarrow A_i^*$ is in the graph, then the conditional distribution $P(A_i^* | A_i, C)$ is determined as outlined above; it is 1 if A_i is in the range defined by the value of A_i^* and 0 otherwise.

If $A_i^* \rightarrow A_i$ is in the graph, then we require that $P(A_i | A_i^*, C) = 0$ whenever A_i is not in the range specified by A_i^* . By Bayes rule $P(A_i^* | A_i) \propto \sum_C P(A_i | A_i^*, C) P(A_i^*, C)$; Thus, if A_i is not in the range of A_i^* , then $P(A_i^* | A_i) \propto \sum_C 0 \times P(A_i^*, C) = 0$. Since the conditional probability of A_i^* given A_i must sum to 1, we conclude that $P(A_i^* | A_i) = 1$ iff A_i is in the range of A_i^* .

There is still the question of the form of $P(A_i | A_i^*, C)$. Our proposal is to learn a model for A_i given A_i^* and C , using the standard methods above (i.e., a Gaussian or a mixture of Gaussians). We then truncate the resulting density on the boundaries of the region specified by the discretization, and we ensure that the truncated density has total mass 1 by applying a normalizing constant. In other words, we learn an unrestricted model, and then condition on the fact that A_i can only take values in the specified interval.

Our goal is then to learn a TAN model that includes both the continuous and discretized versions of each continuous attribute, and that satisfies the restrictions we just described. Since these restrictions are not enforced by the procedure of Section 3.1, we need to augment it. We start by observing that our restrictions imply that if we include $B \rightarrow A$ in the model, we must also include $A \rightarrow A^*$. To see this, note that since A already has one parent (B) it cannot have additional parents. Thus, the only way of making A and A^* adjacent is by adding the edge $A \rightarrow A^*$. Similarly, if we include the edge $B \rightarrow A^*$, we must also include $A^* \rightarrow A$.

This observation suggests that we consider edges between groups of variables, where each group contains both versions of an attribute. In building a TAN structure that includes both representations, we must take into account that adding an edge to an attribute in a group, immediately constraints the addition of other edges within the group. Thus, the TAN procedure should make choices at the level groups. Such a procedure, which we call *hybrid-TAN* is described next.

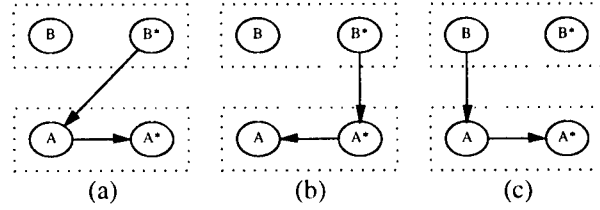


Figure 2: The three possible ways of placing an edge from $\{B, B^*\}$ into $\{A, A^*\}$. The parameterization of possible arcs are as follows: $B^* \rightarrow A^*$ is a discrete model, both $B^* \rightarrow A$ and $B \rightarrow A$ are continuous models (e.g., Gaussians), $A^* \rightarrow A$ is a truncated continuous model (e.g., truncated Gaussian), and $A \rightarrow A^*$ is a deterministic model.

5.2 HYBRID-TAN

We now expand on the details of the procedure. As with the basic procedure, we compute scores on edges. Now, however, edges are between groups of attributes. Each group consisting of the different representations of an attribute.

Let A be a continuous attribute. By our restriction, either $A \in \text{Pa}(A^*)$, or $A^* \in \text{Pa}(A)$. And since each attribute has at most one parent (in addition to the class C), we have that at most one other attribute is in $\text{Pa}(A) \cup \text{Pa}(A^*) - \{A, A^*, C\}$. We define a new function $T(A | B)$ that denotes the best combination of parents for A and A^* such that either B or B^* is a parent of one of these attributes. Similarly, $T(A | \emptyset)$ denotes the best configuration such that no other attribute is a parent of A or A^* .

First, consider the term $T(A | \emptyset)$. If we decide that neither A nor A^* have other parents, then we can freely choose between $A \rightarrow A^*$ and $A^* \rightarrow A$. Thus

$$T(A | \emptyset) = \max(S(A | C, A^*) + S(A^* | C), S(A | C) + S(A^* | C, A)),$$

where $S(A | C, A^*)$ and $S(A^* | C, A)$ are the scores of the CPDs subject to the constraints discussed in Subsection 5.1 (the first is a truncated model, and the second is a deterministic model).

Next, consider the case that a continuous attribute B is a parent of A . There are three possible ways of placing an edge from the group $\{B, B^*\}$ into the group $\{A, A^*\}$. These cases are shown in Figure 2. (The fourth case is disallowed, since we cannot have an edge from the continuous attribute, B to the discrete attribute, A^* .) It is easy to verify that in any existing TAN network, we can switch between the edge configurations of Figure 2 without introducing new cycles. Thus, given the decision that the group B points to the group A , we would choose the configuration with maximal score:

$$T(A | B) = \max(S(A | C, B^*) + S(A^* | C, A), S(A | C, A^*) + S(A^* | C, B^*), S(A | C, B) + S(A^* | C, A))$$

Finally, when B is discrete, then $T(A | B)$ is the maximum between two options (B as a parent of A or as a parent of B^*), and when A is discrete, then $T(A | B)$ is equal to one term (either $S(A | C, B)$ or $S(A | C, B^*)$, depending on B 's type).

We now define the *Hybrid-TAN* procedure:

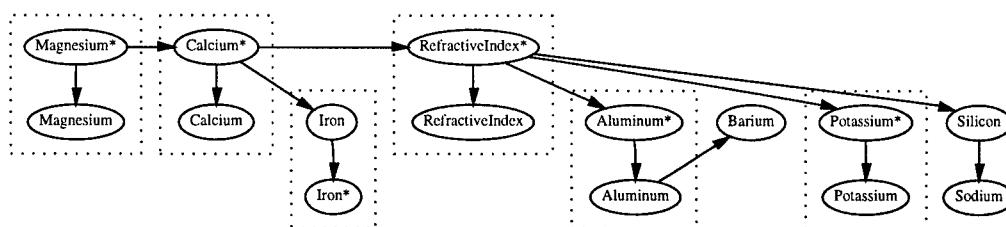


Figure 3: A hybrid TAN model learned for the data set “glass2.” For clarity, the edges from the class to all the attributes are not shown. The attributes marked with asterisks (*) correspond to the discretized representation. Dotted boxes mark two versions of the same attribute.

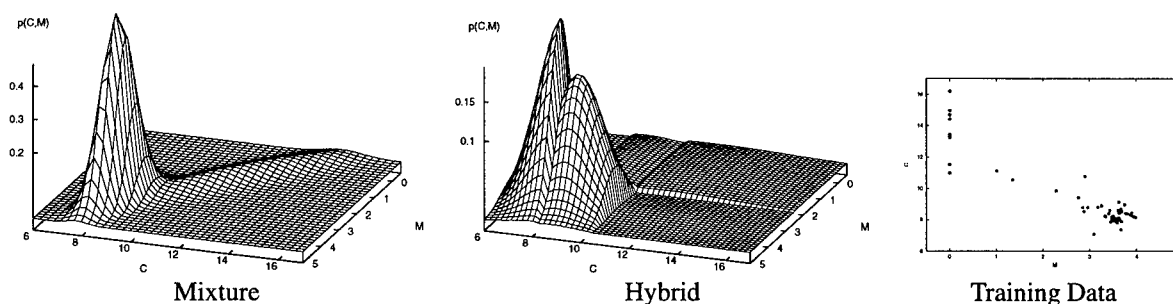


Figure 4: Differences in the modeling of the interaction between attributes, for mixtures of Gaussians and the hybrid model. The graphs show the interaction between Calcium (C) and Magnesium (M) in the “glass2” data set, given a specific value of the class.

1. Initialize an empty graph \mathcal{G} with n vertices labeled $1, \dots, n$.
2. For each attribute A_i , compute the scores of the form $S(A_i | C)$, $S(A_i^* | C)$, $S(A_i | C, A_i^*)$, etc. For each A_j with $j \neq i$, add to \mathcal{G} an arc $j \rightarrow i$ with weight $T(A_i | A_j) - T(A_i | \emptyset)$.
3. Find a maximal weighted branching \mathcal{A} in \mathcal{G} .
4. Construct the TAN model that contains edges from C to each A_i and A_i^* . If $j \rightarrow i$ is in \mathcal{A} , add the best configuration of edges (and the corresponding CPDs) from the group A_j into A_i . If i does not have an incoming arc in \mathcal{A} , then add the edge between A_i and A_i^* that maximizes $T(A_i : \emptyset)$.

It is straight forward to verify that this procedure performs the required optimization:

Theorem 5.1: *The procedure Hybrid-TAN constructs in polynomial time a dual TAN network B that maximizes $LL(B : D)$, given the constraints on the CPDs and the constraint that A_i and A_i^* are adjacent in the graph.*

5.3 AN EXAMPLE

Figure 3 shows an example of a hybrid TAN model learned from one of the folds of the “glass2” data set.¹ It is instructive to compare it to the network in Figure 1, which was learned by a TAN classifier based on mixtures of Gaussians from the same data set. As we can see, there are some similarities between the networks, such as the connections between “Silicon” and “Sodium,” and between “Calcium” and “Magnesium” (which was reversed in the hybrid version). However, most of the network’s structure is quite

¹Some of the discrete attributes do not appear in the figure, since they were discretized into one bin.

different. Indeed, the relation between “Magnesium” and “Calcium” is now modulated by the discretized version of these variables. This fact, and the increased accuracy of hybrid TAN for this data set (see Table 1), seem to indicate that in this domain attributes are not modeled well by Gaussians.

As a further illustration of this, we show in Figure 4 the estimate of the joint density of “Calcium” and “Magnesium” in both networks (given a particular value for the class), as well as the training data from which both estimates were learned. As we can see, most of the training data is centered at one point (roughly, when $M = 3.5$ and $C = 8$), but there is fair dispersion of data points when $M = 0$. In the Gaussian case, C is modeled by a mixture of two Gaussians (centered on 8.3 and 11.8, where the former has most of the weight in the mixture), and M is modeled as a linear function of C with a fixed variance. Thus, we get a sharp “bump” at the main concentration point on the low ridge in Figure 4a. On the other hand, in the hybrid model, for each attribute, we model the probability in each bin by a truncated Gaussian. In this case, C is partitioned into three bins and M into two. This model results in the discontinuous density function we see in Figure 4b. As we can see, the bump at the center of concentration is now much wider, and the whole region of dispersion corresponds to a low, but wide, “tile” (in fact, this tile is a truncated Gaussian with a large variance).

6 EXPERIMENTAL EVALUATION

We ran our experiments on the 23 data sets listed in Table 1. All of these data sets are from the UCI repository [16], and are accessible at the MLC++ ftp site. The accuracy of each classifier is based on the percentage of successful pre-

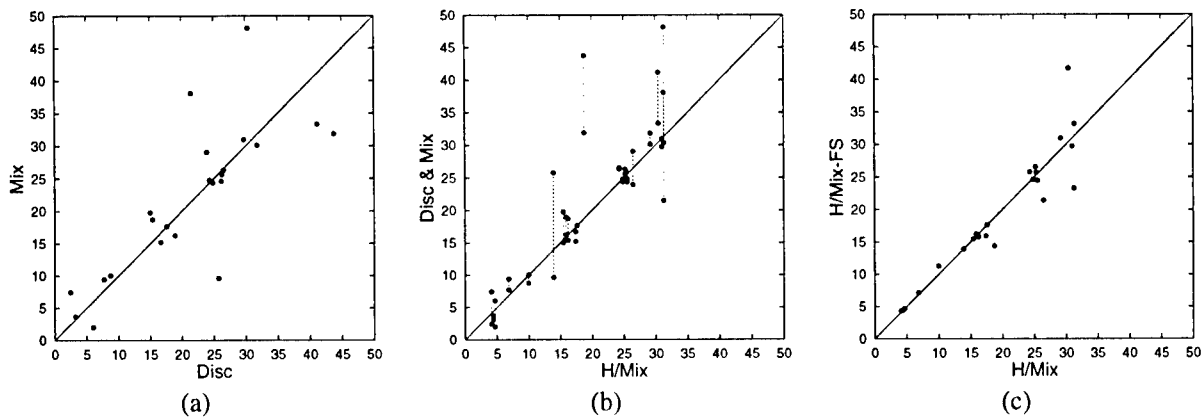


Figure 5: Scatter plots comparing the performance (a) of **Disc** (x axis) vs. **Mix** (y axis), (b) of **H/Mix** (x axis) vs. **Disc** and **Mix** (y axis), and (c) of **H/Mix** (x axis) vs. **H/Mix-FS** (y axis). In these plots, each point represents a data set, and the coordinates correspond to the prediction error of each of the methods compared. Points below the diagonal line correspond to data sets where the y axis method is more accurate, and points above the diagonal line correspond to data sets where the x axis method is more accurate. In (b), the dashed lines connect points that correspond to the same data set.

dictions on the test sets of each data set. We estimate the prediction accuracy of each classifier as well as the variance of this accuracy by using the MLC++ system [14]. Accuracy was evaluated using 5-fold *cross validation* (using the methods described in [13]). Since we do not currently deal with missing data, we removed instances with missing values from the data sets. To construct discretizations, we used a variant of the method of Fayyad and Irani [7], using only the training data, in the manner described in [4]. These preprocessing stages were carried out by the MLC++ system. We note that experiments with the various learning procedures were carried out on exactly the same training sets and evaluated on exactly the same test sets.

Table 1 summarizes the accuracies of the learning procedures we have discussed in this paper: (1) **Disc**—TAN classifier based on prediscritized attributes; (2) **Gauss**—TAN classifier using Gaussians for the continuous attributes and multinomials for the discrete ones; (3) **Mix**—TAN classifier using mixtures of Gaussians for the continuous attributes; (4) **H/Gauss**—hybrid TAN classifier enabling the dual representation and using Gaussians for the continuous version of the attributes; (5) **H/Mix**—hybrid TAN classifier using mixtures of Gaussian for the continuous version of the attributes; and (6) **H/Mix-FS**—same as **H/Mix** but incorporating a primitive form of feature selection. The discretization procedure often removes attributes by discretizing them into one interval. Thus, these attributes are ignored by the discrete version of TAN. **H/Mix-FS** imitate this feature selection by also ignoring the continuous version of the attributes removed by the discretization procedure.

As we can see in Figure 5(a), neither the discrete TAN (**Disc**) nor the mixture of Gaussians TAN (**Mix**) outperforms the other. In some domains, such as “anneal-U” and “glass,” the discretized version clearly performs better; in others, such as “balance-scale,” “hayes-roth,” and “iris,” the semiparametric version performs better. Note that the latter three data sets are all quite small. So, a reasonable hypothesis is that the data is too sparse to learn good discretizations.

On the other hand, as we can see in Figure 5(b), the hybrid method performs at roughly the same level as the best of either **Mix** or **Disc** approaches. In this plot, each pair of connected points describes the accuracy results achieved by **Disc** and **Mix** for a single data set. Thus, the best accuracy of these two methods is represented by the lower point on each line. As we can see, in most data sets the hybrid method performs roughly at the same level as these lower points. In addition, in some domains such as “glass2,” “hayes-roth,” and “hepatitis” the ability to model more complex interactions between the different continuous and discrete attributes results in a higher prediction accuracy. Finally, given the computational cost involved in using EM to fit the mixture of Gaussians we include the accuracy of **H/Gauss** so that the benefits of using a mixture model can be evaluated. At the same time, the increase in prediction accuracy due to the dual representation can be evaluated by comparing to **Gauss**.

Due to the fact that **H/Mix** increases the number of parameters that need to be fitted, feature selection techniques are bound to have a noticeable impact. This is evident in the results obtained for **H/Mix-FS** which, as mentioned above, supports a primitive form of feature selection (see Figure 5(c)). These results indicate that we may achieve better performance by incorporating a feature selection mechanism into the classifier. We leave this as a topic for future research.

7 CONCLUSIONS

The contributions of this work are twofold. First, we extend the TAN classifier to directly model continuous attributes by parametric and semiparametric methods. We use standard procedures to estimate each of the conditional distributions, and then combine them in a structure learning phase by maximizing the likelihood of the TAN model. The resulting procedure preserves the attractive properties of the original TAN classifier—we can learn the best model in polynomial time. Of course, one might extend TAN to use other para-

Table 1: Experimental Results. The first four columns describe the name of the data sets, the number of continuous and discrete attributes, and the number of instances. The remaining columns report percentage classification error and std. deviations from 5-fold cross validation of the tested procedures (see text).

Data set	Attr.			Prediction Errors					
	C	D	Size	Disc	Gauss	Mix	H/Gauss	H/Mix	H/Mix-FS
anneal-U	6	32	898	2.45 ± 1.01	23.06 ± 3.49	7.46 ± 3.12	10.91 ± 1.79	4.12 ± 1.78	4.34 ± 1.43
australian	6	8	690	15.36 ± 2.37	23.77 ± 3.26	18.70 ± 4.57	17.10 ± 2.83	16.23 ± 2.38	15.80 ± 1.94
auto	15	10	159	23.93 ± 8.57	28.41 ± 10.44	29.03 ± 10.04	27.10 ± 8.12	26.47 ± 8.44	21.41 ± 4.27
balance-scale	4	0	625	25.76 ± 7.56	11.68 ± 3.56	9.60 ± 2.47	11.84 ± 3.89	13.92 ± 2.16	13.92 ± 2.16
breast	10	0	683	3.22 ± 1.69	5.13 ± 1.73	3.66 ± 2.13	3.22 ± 1.69	4.34 ± 1.10	4.32 ± 0.96
cars1	7	0	392	26.52 ± 2.64	25.03 ± 7.11	26.30 ± 4.44	25.28 ± 6.54	24.27 ± 7.85	25.79 ± 6.21
cleve	6	7	296	18.92 ± 1.34	17.23 ± 1.80	16.24 ± 3.97	16.24 ± 3.97	15.89 ± 3.14	16.23 ± 3.58
crx	6	9	653	15.01 ± 1.90	24.05 ± 4.44	19.76 ± 4.04	17.31 ± 1.60	15.47 ± 1.87	15.47 ± 2.09
diabetes	8	0	768	24.35 ± 2.56	25.66 ± 2.70	24.74 ± 3.74	22.65 ± 3.21	24.86 ± 4.06	24.60 ± 3.45
echocardiogram	6	1	107	31.82 ± 10.34	28.23 ± 13.86	30.13 ± 14.94	29.18 ± 14.05	29.18 ± 14.05	30.95 ± 11.25
flare	2	8	1066	17.63 ± 4.19	17.91 ± 4.34	17.63 ± 4.46	17.91 ± 4.34	17.63 ± 4.46	17.63 ± 4.19
german-org	12	12	1000	26.30 ± 2.59	25.30 ± 2.97	25.60 ± 1.39	25.70 ± 3.47	25.20 ± 1.75	26.60 ± 2.27
german	7	13	1000	26.20 ± 4.13	25.20 ± 2.51	24.60 ± 1.88	25.10 ± 2.07	25.30 ± 3.33	25.70 ± 4.40
glass	9	0	214	30.35 ± 5.58	49.06 ± 6.29	48.13 ± 8.12	32.23 ± 4.63	31.30 ± 5.00	33.16 ± 5.65
glass2	9	0	163	21.48 ± 3.73	38.09 ± 7.92	38.09 ± 7.92	34.39 ± 9.62	31.27 ± 9.63	23.30 ± 6.22
hayes-roth	4	0	160	43.75 ± 4.42	33.12 ± 11.40	31.88 ± 6.01	29.38 ± 10.73	18.75 ± 5.85	14.38 ± 4.19
heart	13	0	270	16.67 ± 5.56	15.56 ± 5.65	15.19 ± 5.46	15.19 ± 3.56	17.41 ± 4.65	15.93 ± 5.34
hepatitis	6	13	80	8.75 ± 3.42	12.50 ± 4.42	10.00 ± 3.42	12.50 ± 7.65	10.00 ± 5.59	11.25 ± 5.23
ionosphere	34	0	351	7.70 ± 2.62	9.13 ± 3.31	9.41 ± 2.98	6.85 ± 3.27	6.85 ± 3.27	7.13 ± 3.65
iris	4	0	150	6.00 ± 2.79	2.00 ± 2.98	2.00 ± 2.98	4.67 ± 1.83	4.67 ± 1.83	4.67 ± 1.83
liver-disorder	6	0	345	41.16 ± 1.94	40.29 ± 5.16	33.33 ± 4.10	36.52 ± 7.63	30.43 ± 5.12	41.74 ± 2.59
pima	8	0	768	24.87 ± 2.82	24.35 ± 1.45	24.35 ± 3.47	22.92 ± 3.96	25.52 ± 2.85	24.48 ± 2.87
post-operative	1	7	87	29.74 ± 13.06	34.38 ± 10.09	30.98 ± 11.64	34.38 ± 10.09	30.98 ± 11.64	29.74 ± 13.06

metric families (e.g., Poisson distributions) or other semi-parametric methods, (e.g., kernel-based methods). The general conclusion we draw from these extensions is that if the assumptions embedded in the parametric forms “match” the domain, then the resulting TAN classifier generalizes well and will lead to good prediction accuracy. We also note that it is straightforward to extend the procedure to select, at learning time, a parametric form from a set of parametric families.

Second, we introduced a new method to deal with different representations of continuous attributes within a single model. This method enables our model learning procedure (in this case, TAN) to automate the decision as to which representation is most useful in terms of providing information about other attributes. As we showed in our experiments, the learning procedure managed to make good decisions on these issues and achieve performance that roughly as good as both the purely discretized and the purely continuous approaches.

This method can be extended in several directions. For example, to deal with several discretizations of the same attributes in order to select the granularity of discretization that is most useful for predicting other attributes. Another direction involves adapting the discretization to the particular edges that are present in the model. As argued Friedman and Goldszmidt [9], it is possible to discretize attributes to gain the most information about the neighboring attributes. Thus, we might follow the approach in [9] and iteratively readjust the structure and discretization to improve the score. Finally, it is clear that this hybrid method is applicable not only to classification, but also to density estimation and related tasks using general Bayesian networks. We are currently pursuing these directions.

Acknowledgments

We thank Anne Urban for help with the experiments. M. Goldszmidt and T. Lee were supported in part by DARPA's High

Performance Knowledge Bases program under SPAWAR contract N66001-97-C-8548. N. Friedman was supported in part by ARO under grant DAAH04-96-1-0341.

References

- [1] C. M. Bishop. *Neural Networks for Pattern Recognition*, 1995.
- [2] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info. Theory*, 14:462–467, 1968.
- [3] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, 1991.
- [4] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *ICML '95*, 1995.
- [5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*, 1973.
- [6] S. Even. *Graph Algorithms*, 1979.
- [7] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI '93*, 1993.
- [8] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.
- [9] N. Friedman and M. Goldszmidt. Discretization of continuous attributes while learning Bayesian networks. In *ICML '96*, 1996.
- [10] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*, 1995.
- [11] D. Heckerman and D. Geiger. Learning Bayesian networks: a unification for discrete and Gaussian domains. In *UAI '95*, 1995.
- [12] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.
- [13] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI '95*, 1995.
- [14] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. MLC++: A machine learning library in C++. In *Proc. 6th Inter. Conf. on Tools with AI*, 1994.
- [15] S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17:31–57, 1989.
- [16] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. <http://www.ics.uci.edu/~mllearn/MLRepository.html>, 1995.
- [17] R. Tarjan. Finding optimal branching. *Networks*, 7:25–35, 1977.

The Kernel-Adatron Algorithm: a Fast and Simple Learning Procedure for Support Vector Machines

Thilo-Thomas Frieß

Dept. of Automatic Control
and Systems Engineering
University of Sheffield, UK
friess@acse.shef.ac.uk

Nello Cristianini

Dept. of Engineering Mathematics
University of Bristol, UK
nello.cristianini@bristol.ac.uk

Colin Campbell

Dept. of Engineering Mathematics
University of Bristol, UK
c.campbell@bristol.ac.uk

Abstract

Support Vector Machines work by mapping training data for classification tasks into a high dimensional feature space. In the feature space they then find a maximal margin hyperplane which separates the data. This hyperplane is usually found using a quadratic programming routine which is computationally intensive, and is non trivial to implement. In this paper we propose an adaptation of the Adatron algorithm for classification with kernels in high dimensional spaces. The algorithm is simple and can find a solution very rapidly with an exponentially fast rate of convergence (in the number of iterations) towards the optimal solution. Experimental results with real and artificial datasets are provided.

Keywords: Support Vector Machine, Large Margin Classifier, Adatron, Statistical Mechanics

1 INTRODUCTION

Support Vector (SV) machines are an algorithm introduced by Vapnik and co-workers [5, 4] theoretically motivated by VC theory. They are based on the following idea: input points are mapped to a high dimensional feature space, where a separating hyperplane can be found. The algorithm is chosen in such a way to maximize the distance from the closest patterns, a quantity which is called the margin.

This is achieved by reducing the problem to a quadratic programming problem, which is then usually solved with optimization routines from numerical libraries. This step is computational intensive, can be

subject to stability problems and it is non trivial to implement.

SV machines have a proven impressive performance on a number of real world problems such as optical character recognition and face detection [5, 6, 19, 17]. However, their uptake has been limited in practice because of the mentioned problems with the current training algorithms.

An analogous problem has been studied in the Statistical Mechanics literature, which has produced a number of perceptron learning procedures aimed at finding maximal margin hyperplanes in the input space [11, 15, 13]. For some of them also theoretical guarantees are provided, as in the case of Adatron [2], where not only the convergence toward the optimal solution has been proved, but also an exponential rate of convergence in the number of iterations.

We propose a "hybrid" algorithm, the Kernel-Adatron (KA), which combines the implementational simplicity of Adatron with the capability of working in nonlinear feature spaces as SV machines do. By introducing Kernels into the algorithm it is possible to maximize the margin in the feature space, which is equivalent to nonlinear decision boundaries in the input space. The algorithm comes with all the theoretical guarantees given by VC theory for large margin classifiers, as well as the convergence properties studied in the Statistical Mechanics literature.

The result is a fast, robust and extremely simple procedure which implements the same ideas and principles as SV machines at much smaller cost. Experimental results are provided which show that indeed the predictive power of our algorithm is equivalent to that of a SV machine. Furthermore, we show that the running time can be orders of magnitude faster.

2 SUPPORT VECTOR MACHINES

Support Vector (SV) machines implement complex (nonlinear) decision rules in terms of hyperplanes in high dimensional spaces and were originally introduced by Vapnik and co-workers [24, 5, 10].

The decision function realized by SV machines can conceptually be described in two steps: first the training points are mapped by a nonlinear function ϕ to a high-dimensional space where they are linearly separable. Then a separating hyperplane is found which maximizes its distance from the training set, called the margin.

Theoretical results exist from VC theory [24, 21], which guarantee that such solution has high predictive power, in the sense that it minimizes an upper bound on the test error (a complete survey covering the generalization power of SV machines can be found in [3]).

Let $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_p, y_p)\}$ be a sample of points $x_i \in \mathcal{X}$ labelled by $y_i \in \{-1, +1\}$.

Consider a hyperplane defined by (w, θ) , where w is a weight vector and θ a threshold value. Let $S = (X, Y)$ a labeled sample of inputs from \mathcal{X} that has empty intersection with the hyperplane, so that

$$\gamma = \min_{x \in X} |\langle x, w \rangle + \theta| > 0$$

We call this distance the *margin* of the hyperplane w with respect to the sample S .

We also say that the hyperplane is in canonical form with respect to the sample if

$$\min_{x \in X} |\langle x, w \rangle + \theta| = 1$$

It is possible to prove that for canonical hyperplanes

$$\gamma = 1/\|w\|_2$$

The following theorem holds:

Theorem: [21] Suppose inputs are drawn independently according to a distribution whose support is contained in a ball in \mathbb{R}^n centered at the origin, of radius R . If we succeed in correctly classifying m such inputs by a canonical hyperplane with $\|w\| = 1/\gamma$ and $|\theta| \leq R$, then with confidence $1 - \delta$ the generalization error will be bounded from above by

$$\epsilon(m, \gamma) = \frac{1}{m} \left(k \log \left(\frac{8em}{k} \right) \log(32m) + \log \left(\frac{8m}{\delta} \right) \right)$$

where $k = \lfloor 577R^2/\gamma^2 \rfloor$.

The quantity which upper bounds the generalization error does not depend on the dimension of the input space, and this is the theoretical reason why SV machines can use high dimensional spaces without overfitting.

Two main ideas (data-dependent representation and kernels) make it possible to efficiently deal with very high dimensional feature spaces.

The first is based on the identity:

$$\sum_{i=1}^N w_i \phi_i(x) + \theta = \sum_{k=1}^p \alpha_k \phi(x_k) \phi(x) + \theta$$

which provides an alternative, data-dependent, representation of the hypothesis itself, and the other is the use of kernels:

$$K(x', x) = \sum_i \phi_i(x') \phi_i(x)$$

which are equivalent to computing the dot product of the images of two vectors in the feature space [1], provided some (nontrivial) conditions are satisfied.

A common choice are Radial Basis Functions (RBF) such as gaussians,

$$K(x, x') = e^{-\|x-x'\|^2/2\sigma^2}$$

or polynomial kernels

$$K(x, x') = (\langle x, x' \rangle + 1)^d$$

which satisfy such conditions.

The use of the kernels instead of the dot product, in the data-dependent representation of the decision function, automatically provides a way to represent hyperplanes in a feature space rather than in the input space, as first described in [1].

The second conceptual step, aimed at finding the large margin hyperplane, is performed in SV machines by transforming the problem into a Quadratic Programming one, subject to linear constraints.

Kuhn-Tucker theory [24] provides the framework under which the problem can be solved and gives the properties of the solution.

In the data-dependent representation, the lagrangian

$$L = \sum_{i=1}^p \alpha_i - 1/2 \sum_{i,j=1}^p \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

has to be maximized with respect to the α_i , subject to the constraints

$$\alpha_i \geq 0 \quad \sum_{i=1}^p \alpha_i y_i = 0$$

This formulation has a number of interesting properties, characterizing the behaviour of the optimal hyperplane.

There is a Lagrange multiplier α_i for each training point. Only the points which lie closest to the hyperplane, (on parallel hyperplanes at distance γ from the optimal one) have $\alpha_i > 0$ and are called *support vectors*. All the others have $\alpha_i = 0$.

This means that in the representation of the solution, only the points which are closest to the hyperplane contribute: in fact they represent the hypothesis itself, (and their number can also be used to give an independent bound on its reliability [3]).

The resulting decision function can be written as:

$$f(x) = \text{sign} \left(\sum_{i \in \text{SV}} y_i \alpha_i^o K(x, x_i) - \theta \right)$$

where α_i^o is the solution of the constrained maximization problem and SV represents the (indexes of) support vectors.

Such a scheme has proved to be very resistant to overfitting in many classification problems [19, 6, 24]. However this scheme is non trivial to implement, and computationally expensive. Furthermore, in some conditions, it can suffer from numerical conditioning problems.

It is interesting to note that other algorithms which were developed with different motivations have been shown to use a similar technique, equivalent to mapping points to a high dimensional feature space and separating them with a large margin hyperplane. This is the case for Adaboost [18], and for Bayesian Classifiers [7] where the margin distribution over all the

training set is used as an estimator, rather than the margin.

This is justified by a theorem from Schapire *et al.* [18] proving that the fraction of training points which are classified with large margin controls the predictive power, and that valid generalization can be guaranteed even when few points lie near the boundary and hence the margin of the sample is small.

3 THE KERNEL-ADATRON (KA) ALGORITHM

In the Statistical Mechanics approach to learning [25], a very similar problem has been studied, with different motivations. The “perceptron with optimal stability” has been the object of extensive theoretical and experimental work, [15, 11, 2], and a number of simple iterative procedures have been proposed, aimed at finding hyperplanes which have “optimal stability” or - in our terms - maximal margin.

One of them, the Adatron, comes with theoretical guarantees of convergence to the optimal solution, and of a rate of convergence exponentially fast in the number of iterations [2, 15], provided that a solution exists.

We demonstrate that such models can be adapted, with the introduction of kernels, to operate in a high-dimensional feature space, and hence to learning non-linear decision boundaries. This provides a procedure which emulates SV machines but doesn't need to use the quadratic programming toolboxes.

In this section we will briefly sketch the Adatron algorithm, and we will list the theoretical results which can be proved for it (in the Statistical Mechanics framework), pointing to the relevant papers for the proofs of the theorems. Finally we will show how it is possible to introduce the kernels. The next section will be devoted to experimental comparisons between KA and SV machine, and to benchmarking.

The Adatron is a an on-line algorithm for learning perceptrons which has an attractive fixed point corresponding to the maximal-margin consistent hyperplane, when this exists.

By writing the Adatron in the data-dependent representation, and by substituting the dot products with kernels, we obtain the following algorithm:

The Kernel-Adatron Algorithm.

1. Initialise $\alpha_i = 1$.
2. Starting from pattern $i = 1$, for labeled points (x_i, y_i) calculate $z_i = y_i \sum_{j=1}^p \alpha_j y_j K(x_i, x_j)$.
3. For all patterns i calculate $\gamma_i = y_i z_i$ and execute steps 4 to 5 below.
4. Let $\delta\alpha^i = \eta(1 - \gamma^i)$ be the proposed change to the multipliers α^i .
- 5.1. If $(\alpha^i + \delta\alpha^i) \leq 0$ then the proposed change to the multipliers would result in a negative α^i . Consequently to avoid this problem we set $\alpha^i = 0$.
- 5.2 If $(\alpha^i + \delta\alpha^i) > 0$ then the multipliers are updated through the addition of the $\delta\alpha^i$ i.e. $\alpha^i \leftarrow \alpha^i + \delta\alpha^i$.
6. Calculate the bias b from

$$b = \frac{1}{2} (\min(z_i^+) + \max(z_i^-))$$

where z_i^+ are those patterns i with class label +1 and z_i^- are those with class label -1.

7. If a maximum number of presentations of the pattern set has been exceeded then **stop**, otherwise return to step 2.

The kernel $K(x, x')$ can be any function satisfying Mercer's condition, in particular it is possible to use RBF or polynomial kernels given in section 2.

Important Remarks

Using results reported in the Statistical Mechanics literature, the following important properties of the Adatron can be derived:

1. (Anlhauf and Biehl [2]) *Every stable point for the Adatron algorithm is a maximal margin point and vice versa.*

Proof Sketch

By inserting the Kuhn-Tucker conditions for the maximal margin ($\alpha_i > 0 - \gamma_i = 1$, $\alpha_i = 0 - \gamma_i > 1$) in the Adatron updating rule it follows that the optimal margin is a fixed point. Vice versa by imposing $\delta\alpha_i = 0 \forall i$ the Kuhn-Tucker conditions are obtained.

2. (Anlhauf and Biehl [2]) *The algorithm converges in a finite number of steps to a stable point if a solution exists.*

Proof Sketch The functional

$$L = \sum_{i=1}^p \alpha_i - 1/2 \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

can be shown to be upper bounded, and to increase monotonically at each updating step of the Adatron. So it has to find a fixed point in a finite number of steps.

3. (Oppner [16], [15]) *The rate of convergence to the optimal solution follows an exponential law in the number of iterations.*

The proof makes use of replica calculations from Statistical Mechanics (and the standard assumptions of that model [25]).

Note: The convergence proof relies on an adequate choice of η , which also controls the speed of the convergence itself. The issues regarding the choice of η cannot be discussed here for lack of space, but we observe that the theory provides an interval within which a valid η can be chosen. Results will be presented elsewhere.

4 EXPERIMENTAL RESULTS

We have evaluated the performance of the KA algorithm with gaussian kernels on a number of standard classification datasets, both artificial and real. The artificial datasets include the two-spirals problem [8], n-parity [14], mirror symmetry [14]. The real world data include the sonar classification problem [9], the Wisconsin breast cancer dataset [23] and a database of handwritten digits collected by the US Postal Service [12].

4.1 THE TWO SPIRALS PROBLEM AND n-PARITY

For the two spirals problem the task is to discriminate between two sets of points which lie in two spirals in a plane.

The solution found by the KA algorithm is illustrated in Figure 2 and compared with the solution provided by a kernel-perceptron, i.e. a generic hyperplane in the feature space (Fig. 1).

The diagrams present different decision functions; in the kernel-perceptron's case the small margin yields a highly non smooth boundary while for the KA algorithm a smooth and centered solution has been found.

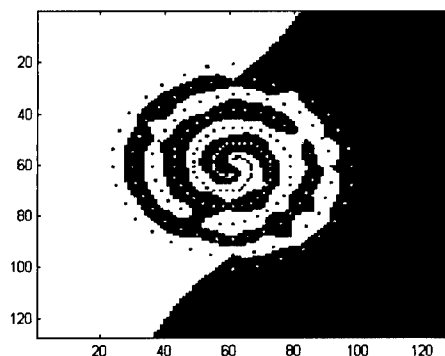


Figure 1: Kernel-Perceptron (small margin) clearly overfits

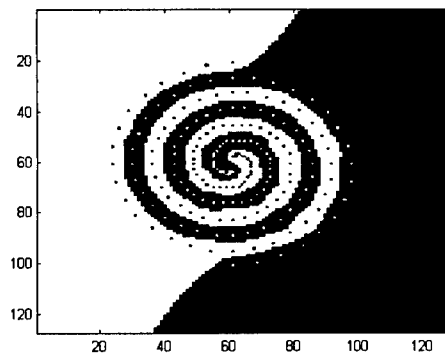


Figure 2: Kernel-Adatron learns a much smoother boundary

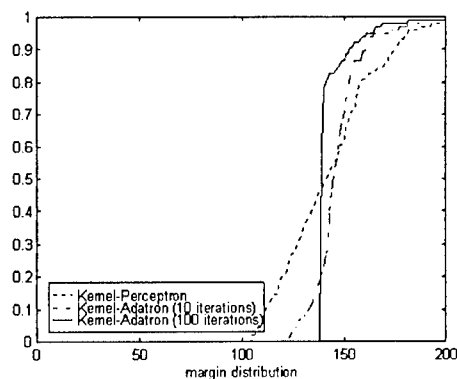


Figure 3: Cumulative margin distribution for kernel-perceptron and for KA. Note the scaling of the margins: we denote with 100 the null margin (points on the boundary).

Useful insight about the differences between these two learning machines can be obtained by observing the margin distribution graphs in Fig. 3, which present the cumulative distribution of the margins of all individual training points, i.e. the fraction of patterns (vertical axis) which have a margin larger than a given value θ (horizontal axis). It is interesting to note that the effect on the margin distribution of the training in KA is similar to the one in Adaboost, discussed in [18].

The solution for the n -parity problem [14], which is hard to separate for neural networks, was found in 1 epoch for $n = 3$ and $n = 6$, while it took respectively 3 and 5 epochs to maximise the margin.

4.2 MIRROR SYMMETRY

In the mirror symmetry problem [14] the output y is a 1 if the input pattern x (with components from $\{-1, +1\}$) is exactly symmetrical about its centre, otherwise the output is a -1 . For randomly constructed input strings the output would be a -1 with a high probability. Consequently the labels ± 1 are selected with a 50% probability and the first half of the input string is randomly constructed from components in $\{-1, +1\}$ (both selected with a 50% probability) and the second half of the string is symmetrical or random depending on the target value given. Generalisation was evaluated using a test set drawn from the same distribution (eliminating any instances for which the input string is identical to a member of the training set).

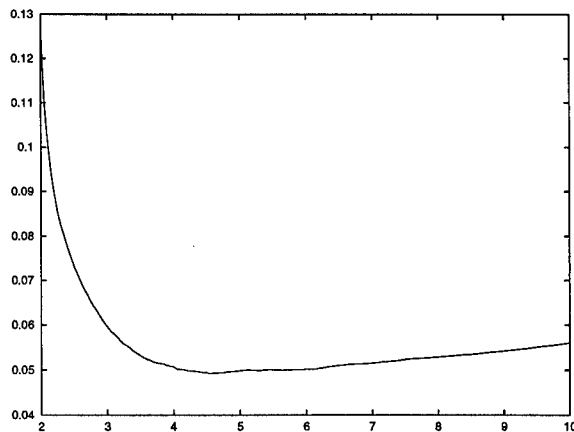
In Figure 4 we plot the generalisation error on the test set (100,000 examples including repetitions) versus σ for the KA algorithm trained to 200 epochs with $\eta = 1.0$. 200 training examples were used with input strings consisting of 30 components. The generalisation error passes through a minimum between $\sigma = 4 - 5$ with a maximum generalisation of 95.1%. To compare with other algorithms in a machine independent way we have implemented all algorithms in MATLAB (using its optimization toolbox) and estimated the individual speeds using FLOPS (Table 1). We see that the KA algorithm is substantially faster than Support Vector machines while also having a comparable generalisation performance to the latter (TR is the number training errors, TS the number of test errors on a set of 100 patterns). It also performs much better than k -nearest neighbour (k NN) on the test set.

Alg.	EP.	σ	TR.	TS.	FLOPS
kNN(k=1)	-	-	0	25	0
kNN(k=7)	-	-	0	22	0
SVM	-	3.5	0	3	0.173×10^9
SVM	-	4.2	0	2	0.318×10^9
SVM	-	5.0	0	5	0.694×10^9
KA	10	3.5	0	4	1210000
KA	10	4.2	0	5	1210000
KA	10	5.0	0	6	1210000
KA	100	3.5	0	3	12100000
KA	100	4.2	0	3	12100000
KA	100	5.0	0	4	12100000
KA	250	3.5	0	3	30250000
KA	250	4.2	0	4	30250000
KA	250	5.0	0	5	30250000

Table 1: comparison for mirror symmetry

4.3 SONAR CLASSIFICATION

The sonar classification problem of Gorman and Sejnowski [9] consists of 208 instances formed by 60 analogue inputs, representing returns from a roughly cylindrical rock or a metal cylinder, equally divided into training and test sets. For the aspect-angle dependent dataset [9] they trained a standard back-propagation neural network with 60 inputs and 2 output nodes. Experiments were performed with up to 24 hidden nodes and each neural network was trained with 300 epochs through the training set. Their results are reproduced in Table 2.

Figure 4: Generalisation error (vertical axis) vs. σ (horizontal axis): mirror symmetry problem

# hidden	0	2	3	6	12	24
% gen.	73.1	85.7	87.6	89.3	90.4	89.2

Table 2: Gorman and Sejnowski results for sonar

For the KA algorithm we plot σ against generalisation

error in Figure 5 and the best generalisation performance is 95.2% by comparison. The KA algorithm is also very fast. Figure 6 illustrates the approach of the margin towards 1 (for $\sigma = 1.0$ and $\eta = 1.0$). The training error fell to 0 in the second epoch (it was 0.077 at the end of the first epoch). We also show the generalisation error versus number of epochs (Figure 7). As for mirror symmetry we give a comparison with Support Vector Machines in Table 3.

Alg.	EP.	σ	TR.	TS.	FLOPS
kNN(k=1)	-	-	0	10	0
kNN(k=3)	-	-	0	19	0
SVM	-	0.57	0	8	3.476×10^9
SVM	-	0.71	0	7	6.750×10^9
SVM	-	0.85	0	7	8.878×10^9
KA	10	0.57	0	6	329680
KA	10	0.71	0	9	329680
KA	10	0.85	0	8	329680
KA	100	0.57	0	6	3296800
KA	100	0.71	0	6	3296800
KA	100	0.85	0	7	3296800
KA	250	0.57	0	6	8242000
KA	250	0.71	0	6	8242000
KA	250	0.85	0	7	8242000

Table 3: comparison for sonar classification

4.4 WISCONSIN BREAST CANCER DATASET

The Wisconsin breast cancer dataset contains 699 patterns with 10 attributes for a binary classification task (the tumour is malignant or benign).

This dataset has been extensively studied by other authors. CART gives a generalisation of 94.2%, an RBF neural network gave 95.9%, a linear discriminant method gave 96.0% and a multi-layered neural network (trained via Back-Propagation) 96.6% (all the results have been obtained using 10-fold cross-validation [23]). Our optimal test performance was of 99.48%, which is superior to the previous reported results. However we regard this result as simply indicating that we are comparable with other approaches, as this difference can also be due to other factors and requires further investigation. Among them are differences in the handling of instances with missing values (16 in the database), in the preprocessing (we have removed the first column of the database reporting the patient's code number, like some other authors) and in the choice of σ . We note that the test error is insensitive to the choice of σ in a broad interval, as can be seen in Fig. 11. In this diagram we give a plot of generalisation error versus σ for 10-fold cross validation on the 699 instances (50 iterations were used and $\eta = 1.0$).

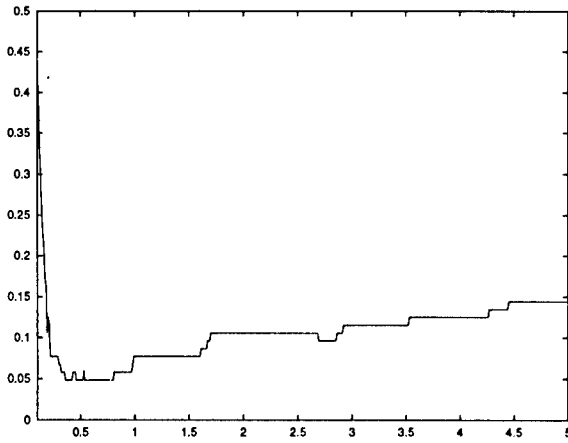


Figure 5: Generalisation error of KA (vertical axis) vs. σ (horizontal axis) for the sonar classification problem.

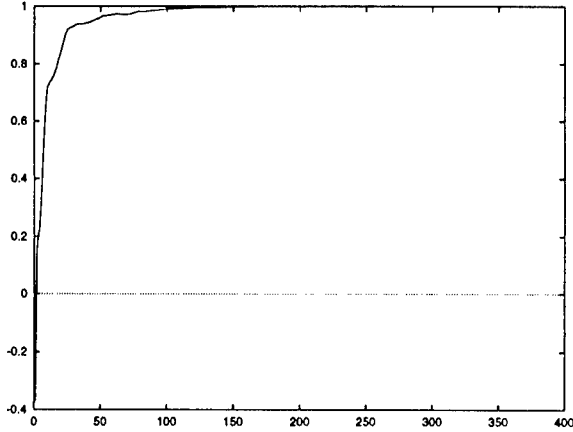


Figure 6: Margin (vertical axis) vs. number of epochs (horizontal axis) for sonar classification ($\sigma = 1.0$, $\eta = 1.0$).

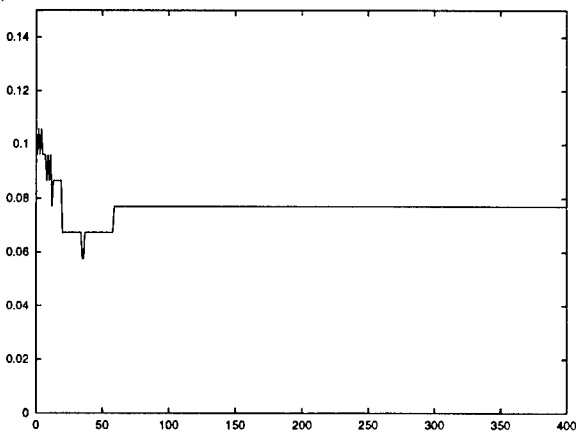


Figure 7: Generalization error (vertical axis) vs. number of epochs (horizontal axis) for sonar classification ($\sigma = 1.0$, $\eta = 1.0$).

Furthermore, for a particular split of the database with 550 training examples and 149 test examples, $\sigma = 3.2$ and $\eta = 1.0$, we give plots of the generalisation error (Fig. 8), margin (Fig. 9) (all versus number of epochs and the final spectrum of α values (Figure 10).

To compare the computational cost of KA with other classifiers we have used a matlab implementation of them, and run it on a reduced subset of the database (199 training and 168 testing points) using the FLOPS as an indication of the algorithmic complexity. The results are reported in Table 4 and indicate that KA can achieve about the same generalization performance of SV machines at a cost which is orders of magnitude smaller.

Alg.	EP.	σ	TR.	TS.	FLOPS
kNN(k=1)	-	-	0	13	0
kNN(k=3)	-	-	0	9	0
SVM	-	0.28	0	11	2.4541×10^9
SVM	-	0.35	0	10	2.7763×10^9
SVM	-	0.42	0	11	2.8043×10^9
KA	10	0.28	0	8	1197980
KA	10	0.35	0	9	1197980
KA	10	0.42	0	11	1197980
KA	100	0.28	0	10	11979800
KA	100	0.35	0	9	11979800
KA	100	0.42	0	9	11979800
KA	250	0.28	0	10	29949500
KA	250	0.35	0	9	29949500
KA	250	0.42	0	10	29949500

Table 4: comparison for cancer classification (a subset has been used for this comparison)

4.5 US POSTCODE DATABASE

The benchmarking of classification algorithms of the class of SV machines has traditionally been performed using the database of handwritten digits from US Postal Codes [12, 20].

This dataset consists of a training set of 7,291 examples and a test set of 2,007. Each digit is given by a 16×16 vector with components which lie in the range -1 to 1 . In this experiment we have performed two-class classification i.e. separating a particular digit from the others. To find suitable values for σ the training set was split into a smaller training set of 6,000 examples and a validation set of 1,291. The best value of σ was found by evaluating performance on the validation set across the range $(1, 10)$. The full training set of 7,291 was then used with the selected value of σ to train the system to classify each digit.

The results are shown in Table 5 where the last column shows the best value of σ found from the validation

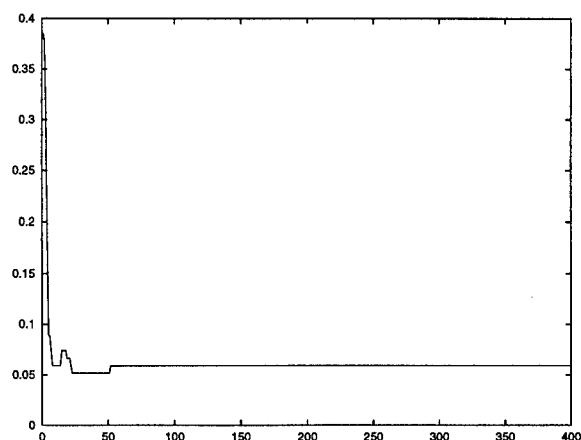


Figure 8: Generalization error (vertical axis) vs. number of epochs (horizontal axis) for cancer classification

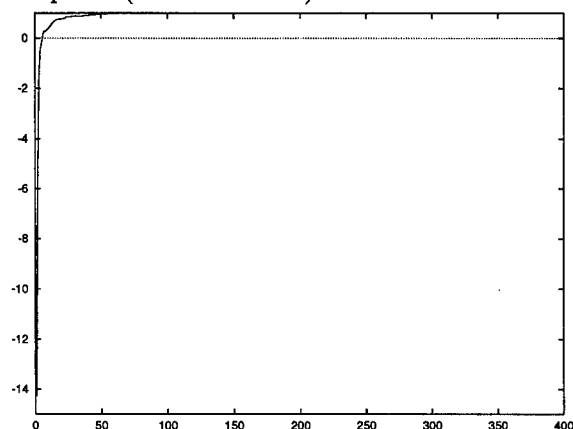


Figure 9: Margin (vertical axis) vs. number of epochs (horizontal axis) for cancer classification

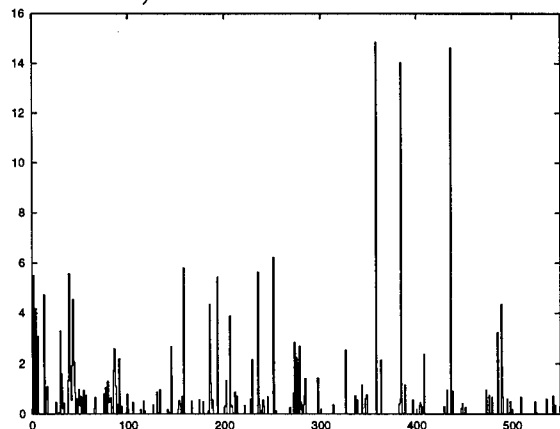


Figure 10: Spectrum of α values for the 550 patterns found by the KA algorithm (cancer classification experiment)

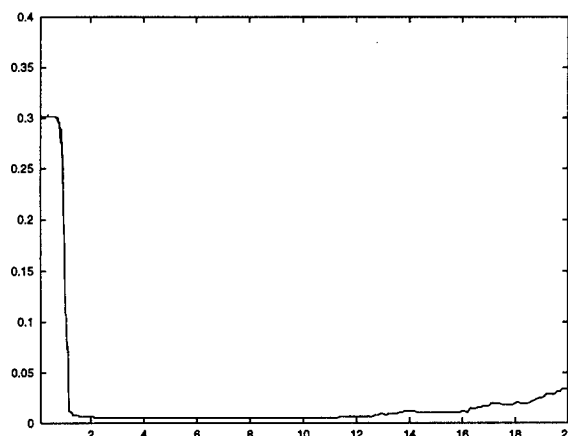


Figure 11: Generalisation error (vertical axis) vs. σ (horizontal axis) for cancer classification ($\eta = 1.0$)

study. The other columns show the number of errors on the test set of 2,007 examples for the KA algorithm and 3 comparative algorithms as reported by [20]. The latter three algorithms are an RBF neural network, a Support Vector Machine (SVM) and a hybrid model in which the support vectors found by the SVM are used as the centers of receptive fields in an RBF network [20].

Digit	RBF	SVM	Hybrid	KA	σ
0	20	16	9	13	1.8
1	16	8	12	10	1.6
2	43	25	27	21	2.4
3	38	19	24	24	2.0
4	46	29	32	26	4.0
5	31	23	24	19	1.8
6	15	14	19	15	2.4
7	18	12	16	11	2.8
8	37	25	26	26	3.2
9	26	16	16	14	1.6

Table 5: comparative performance on the USPS database (number of errors in a 2007 points test set)

The performance of KA is comparable with the other algorithms.

5 CONCLUSIONS AND FUTURE WORK

We have presented an algorithm which finds maximal margin hyperplanes in a high dimensional feature space, emulating Vapnik's Support Vector machines.

Experiments performed on artificial and real data show that the generalization performance of this algorithm

is comparable with that of SV machines, while the computational cost of finding the hypothesis is significantly smaller. Also, the introduction of kernels into the Adatron provides a very simple, compact and robust algorithm.

Further work is now needed to introduce the capability of tolerating training errors, so that the machine can deal with outliers and noisy datasets, following the soft-margin approach used in SV machines.

Acknowledgements This work was partially funded by EPSRC. Nello wishes to thank John Shawe-Taylor and Jason Weston for many useful discussions. Thilo would like to thank Rob Harrison and Klaus-Robert Müller, and Univ. of Sheffield/AC&SE for support.

References

- [1] Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning, *Automations and Remote Control*, **25**:821-837.
- [2] Anlauf, J.K., and Biehl, M. (1989). *Europhysics Letters* **10**:687
- [3] Bartlett P., Shawe-Taylor J., (1998). Generalization Performance of Support Vector Machines and Other Pattern Classifiers. 'Advances in Kernel Methods - Support Vector Learning', Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), MIT Press, Cambridge, USA.
- [4] Boser, B., Guyon, I., Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Fifth Annual Workshop on Computational Learning Theory*. ACM Press.
- [5] Cortes, C., and Vapnik, V. (1995). Support Vector networks, *Machine Learning* **20**:273-297.
- [6] Cortes, C. (1995). *Prediction of Generalization Ability in Learning Machines*. PhD Thesis, Department of Computer Science, University of Rochester.
- [7] Cristianini, N., Shawe-Taylor, J., Sykacek, P., (1998). Bayesian Classifiers are Large Margin Hyperplanes in a Hilbert Space, in Shavlik, J., ed., *Machine Learning: Proceedings of the Fifteenth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA.
- [8] Frieß T.-T., Harrison R.F., (1998), Pattern Classification using Support Vector Machines, Eng. and Int. Sys. (EIS98 Conf. Proc.)
- [9] Gorman R.P. & Sejnowski, T.J. (1988) *Neural Networks* **1**:75-89.
- [10] Guyon, I., Matic, N., & Vapnik, V. (1996). Discovering Informative Patterns and Data Cleaning, *Advances in Knowledge Discovery and Data Mining* ed by U.M.Fayyad, G. Piatelsky-Shapiro, P. Smyth and R. Uthurusamy AAAI Press/ MIT Press.
- [11] Kinzel, W., (1990) Statistical Mechanics of the Perceptron with Maximal Stability. *Lecture Notes in Physics* (Springer-Verlag) **368**:175-188.
- [12] LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P. and Vapnik, V., (1995). Comparison of learning algorithms for handwritten digit recognition, *International Conference on Artificial Neural Networks*, Fogelman, F. and Gallinari, P. (Ed.), pp. 53-60.
- [13] Krauth W. and Mezard. M. (1987) *J.Phys.* **A20**:L745
- [14] Minsky M.L. & Papert, S.A. (1969) *Perceptrons*, MIT Press: Cambridge.
- [15] Oppen, M. (1988). Learning Times of Neural Networks: Exact Solution for a Perceptron Algorithm. *Physical Review* **A38**:3824
- [16] Oppen, M. (1989). Learning in Neural Networks: Solvable Dynamics. *Europhysics Letters*, **8**:389
- [17] Osuna E., Freund R., Girosi F., (1997) "Training Support Vector Machines: An Application to Face Detection", Proc. Computer Vision and Pattern Recognition '97, 130-136
- [18] Schapire. R., Freund, Y., Bartlett, P., & Sun Lee, W. (1997). Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods, *Proceedings of International Conference on Machine Learning*.
- [19] Schoelkopf, B., (1997). *Support Vector Learning*. PhD Thesis. R. Oldenbourg Verlag, Munich.
- [20] Schoelkopf, B., Sung, K., Burges, C., Girosi, F., Niyogi, P., Poggio, T., and Vapnik, V., Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers, M.I.T. Preprint (A.I. Laboratory), A.I. Memo No. 1599.
- [21] Shawe-Taylor, J., Bartlett, P., Williamson, R. & Anthony, M. (1996). Structural Risk Minimization over Data-Dependent Hierarchies NeuroCOLT Technical Report NC-TR-96-053 (<ftp://ftp.dcs.rhnc.ac.uk/pub/neurocolt/tech.reports>).
- [22] (Sonar dataset) <http://www.boltz.cs.cmu.edu/benchmarks/sonar.html>
- [23] Ster, B., & Dobnikar, A. (1996) Neural networks in medical diagnosis: comparison with other methods. In A. Bulsari et al. (ed.) *Proceedings of the International Conference EANN'96*, p. 427-430.
- [24] Vapnik, V. (1995) *The Nature of Statistical Learning Theory*, Springer Verlag.
- [25] Watkin, T., Ran, A. & Biehl, M. (1993). The Statistical Mechanics of Learning a Rule, *Rev. Mod. Phys.* **65**(2).

Multi-criteria Reinforcement Learning

Zoltán Gábor, Zsolt Kalmár and Csaba Szepesvári

Associative Computing Ltd.

Budapest 1121, Konkoly Thege M. út 29-33

e-mails: {gzoli,kalmar,szepes}@mindmaker.kfkipark.hu

Abstract

We consider multi-criteria sequential decision making problems where the vector-valued evaluations are compared by a fixed total ordering of the vectors. Conditions for the optimality of stationary policies and the Bellman optimality equation are given for a special, but important class of problems, when the evaluation of policies can be computed componentwise. The analysis requires special care as the topology introduced by pointwise convergence and the order-topology introduced by the preference order are in general incompatible several. Reinforcement learning algorithms are then proposed and analyzed. Preliminary computer experiments confirm the validity of the derived algorithms. These type of multi-criteria problems are most useful when there are several optimal solutions to a problem and one wants to choose the one among these which is optimal according to another fixed criterion. Possible application in robotics and repeated games are outlined.

1 Introduction

Scalar-valued reinforcement learning (RL) algorithms are capable of solving difficult multi-step decision problems when the decision criteria can be expressed in a recursive way as a function of the immediate scalar reinforcement. However, there are some important cases when there is no simple way to express the optimization criteria as a function of a single scalar reinforcement value. Consider, for example, the dilemma

of Buridan's ass.¹ This poor animal is placed at equal distances away from two platefuls of food. He is hungry so he feels like going to one of the plates. However, if he goes to one plate then there is a chance that the dish from the other one gets stolen. Since the ass is greedy (he does not want any dish to be stolen away) he will never move and will, eventually, die.

In this example the ass has two different objectives competing with one another. The first one is to eat so that he can stay alive, the second one is to prevent the dishes from being stolen. A reasonable compromise, which could be termed the "watchmen's compromise", is to minimize the number of dishes stolen per unit time such that the ass manages to stay alive: $\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^T S_t \rightarrow \min$ s.t. $\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^T R_t \geq R_{\text{crit}}$. Here $S_t \in \{0, 1\}$ is the indicator of whether a plate was stolen at time t , $R_t = \{0, 1\}$ is the indicator of whether the ass was consuming at time t , and R_{crit} is the critical amount of food per unit time needed for staying alive. We can use a Tauberian approximation to the above criterion [Ross, 1970]:

$$\sum_{t=0}^{\infty} \gamma^t S_t \rightarrow \min \quad \text{s.t.} \quad \sum_{t=0}^{\infty} \gamma^t R_t \geq R'_{\text{crit}}, \quad (1)$$

where $0 < \gamma < 1$ is a value sufficiently close to 1,

¹Buridan, a French philosopher of the mediaeval period, wrote several significant commentaries on the classical philosophical, logical, and physical works of Aristotle, including the *Physics*. Actually, he never referred to the infamous ass in his extant writings, but this concept was invented by his opponents to ridicule his use of animals in the examples he used to expound his theories on free will. In the original version of the story a hungry ass stood between two haystacks, both of which were equally appetizing. Unable to decide from which stack to eat, the ass eventually starved to death. However, the example in this form did not serve well our purposes so we felt free to modify it slightly.

$R'_{\text{crit}} = R_{\text{crit}}/(1 - \gamma)$.² Since the decision should be made on the basis of both the amount of food eaten and the number of plates stolen and both of these should be computed separately, the normal form of reinforcement at time t will be (R_t, S_t) .

Note that there exists other ways to formalize the dilemma of Buridan's ass. Another reasonable compromise, e.g., is to maximize the weighted sum of protected plates and the amount eaten: $\sum_{t=0}^{\infty} \gamma^t (w_1(1 - S_t) + w_2 R_t) \rightarrow \max$, where $w_1, w_2 > 0$. This reduces the problem to the case of scalar-valued reinforcement values. Here, we do not want to argue against this or other reductions, but we want to show that under certain conditions reinforcement learning algorithms can be extended to the vector-valued case in a sensible way.

If the immediate reinforcement is vector-valued then so will be the long-term reinforcement, and, specifically, the evaluation of policies. Then the comparison of policies becomes problematic. The requirements for a meaningful comparison are the following: we want to compare any pairs of policies and, in particular, we want a transitive and reflexive comparison operator. Several approaches will be shown below. No matter how the policies are compared the notion of an optimal policy can be defined at this point: an optimal policy is one which compares favorably with any other policy.

The comparison methods are best illustrated by the above problem. Let $v_{\pi}(x) \in \mathbf{R}^2$ denote the evaluation of policy π in state x with $v_{\pi}(x)^T = (v_{\pi,1}(x), v_{\pi,2}(x))$, where $v_{\pi,1}(x)$ is the maximum of the amount of food eaten and R_{crit} , while $v_{\pi,2}(x)$ is the number of plates stolen, both being computed when policy π is being used beginning from state x . The criterion of the introduction suggests to compare any pair of policies (π_1, π_2) by first comparing the first components of their respective evaluation functions: π_1 is better than π_2 if $v_{\pi_1,1}(x) > v_{\pi_2,2}(x)$. Since evaluations are cut at R_{crit} we may expect that $v_{\pi_1,1}(x)$ and $v_{\pi_2,2}(x)$ will be equal in a large number of cases. Then, we compare the second components: π_1 is better than π_2 if $v_{\pi_1,2}(x) < v_{\pi_2,2}(x)$ (note the reversed relational symbol). That is, among policies which let Buridan's ass stay alive, the ones with a smaller number of stolen plates are preferred. Since here the policies are compared on the basis of an *ordering among the vector-components* of the policy evaluation functions, this problem is one example of *ordinal multi-criteria deci-*

sion problems, which were considered a long time ago by Mitten [1964] and Sobel [1975] in terms of *preference* relations over "partial policies". In order the subordinate criteria to be useful at all, the optimization problem corresponding to the main objective should have multiple solutions. This can be achieved using reduced reinforcement-spaces. As an interesting example note that Asimov's robots obey multi-criteria rules of this form. The "laws of robotics" claims that robots have to *i)* defend human beings, *ii)* defend themselves unless this conflicts with rule *i)*; and *iii)* serve human beings unless this conflicts with rules *i)* or *ii)*. This can be clearly understood as an ordinal multi-criteria optimization problem. This type of criterion is also related to solving MDPs in parallel, a problem similar to that of considered by Singh and Cohn [1997] and empirically Asada et al. [1994] for football playing robots. In this latter case a robot's primary goal could be to win the game, while its subordinate goal could be to keep clear of opponents as much as it is possible.

Criterion (1) can also be viewed as one that defines a discounted optimization problem subject to a discounted constraint. Structural properties of such problems were studied extensively in the control and operations research literature, e.g. by Frid [1972], Heyman and Sobel [1984], Altman and Schwartz [1991].

Another approach is to compare any pair of policies, (π_1, π_2) , by comparing the weighted sum of the components of their evaluation functions, e.g. $w_1 v_{\pi_1,1}(x) + w_2 v_{\pi_1,2}(x)$ and $w_1 v_{\pi_2,1}(x) + w_2 v_{\pi_2,2}(x)$ ($w_1, w_2 \in \mathbf{R}$). Note that this criterion, often called the weighted criterion (see Feinberg and Schwartz [1995] and the references therein), is different from the one obtained by the linear combination of the immediate reinforcement values iff the discount factors of the two components are different.

If there is no natural weighing of components then one can still use the canonical ordering over the return space. In this case, however, not all policies will be comparable and so the notion of optimality needs to be adjusted. The natural choice is then *Pareto-optimality*: a policy π is called Pareto-optimal in state x if no other policy can majorize π at x , i.e., if there is no policy π' s.t. $v_{\pi'}(x) \geq v_{\pi}(x)$. A policy is called Pareto-optimal iff it is Pareto-optimal for each state. It turns out, that Pareto-optimality is equivalent to weighted optimality with appropriately chosen weights and if each component of the evaluation is computed as the total discounted reward for some reward function [Feinberg and Schwartz, 1995, Lemma 7.4]. In the above example, assuming that the amount of con-

²In order to simplify the presentation we implicitly assume here that the decision process is deterministic. However, this assumption is in no way essential to the subsequent developments and will be abandoned later.

sumed food is not truncated, a Pareto-optimal policy would be one for which there is no other policy that would allow the ass to consume more (than the amount ensured by the Pareto-optimal policy) while assuring a smaller number of stolen plates at the same time. Pareto-optimality has been studied by many researchers from the point of view of providing conditions which ensure the existence of optimal policies of certain forms (stationary policies are not Pareto-optimal in general).

Apparently the earliest result for dynamic vector-valued models are those of Brown and Strauch [1965], who considered abstract return spaces having a general multiplicative lattice structure and who showed that the "principle of optimality" holds for finite-horizon problems. Their results were later extended to infinite horizon problems in many special cases (see, e.g. [Feinberg, 1982, Henig, 1983, Feinberg and Schwartz, 1994]).

In this article we present a general framework based on abstract dynamic programming models, and which is a mixture of the above approaches [Denardo, 1967, Bertsekas, 1977, Littman and Szepesvári, 1996, Szepesvári, 1998]. Namely, we suggest an approach based on the notion of reinforcement-propagating operators, which now act on function spaces defined over an *abstract return space with a given ordering*. In this way we can address constrained problems, lexicographic criteria, lattice return spaces and different reinforcement propagation scenarios within the same framework.

The article is organized as follows: in Section 2 we introduce the concepts necessary for the development and list some basic results concerning the Bellman-optimality equation and the existence of optimal stationary policies. Reinforcement learning algorithms are introduced in Section 3. Some computer experiments, illustrating the theory, are given in Section 4 and conclusions are drawn in Section 5.

2 Abstract ordinal dynamic programming

An Abstract Dynamic Programming (ADP) problem can be given as a 5-tuple $(\mathcal{R}, X, A, \mathcal{A}, \mathcal{Q})$, where X is the state-space of the decision problem, A is the set of actions, $A : X \rightarrow A$, $\mathcal{A}(x)$ are the actions feasible in state x , \mathcal{R} is the return space and $\mathcal{Q} : \mathcal{R}^X \rightarrow \mathcal{R}^{X \times A}$ is the so-called reinforcement-propagator operator [Szepesvári, 1998].³ In order to explain the

meaning of these components consider the problem of Buridan's ass once again. A simplified representation of that problem could be the following: the ass's state assumes three values: being in the middle, at the plate, or at the right plate. The plates can be empty or full. A state of the decision problem is composed of the position of the ass, and the states of the plates. So the state space (X) has 12 elements. The actions taken by the ass can be to stay at that position, move to the left, or move to the right; so the action space (A) has three elements. The dynamics is given by the following (stochastic) rules: the move actions work as intended. If the ass chooses to stay at a full plate then that plate becomes empty (consuming), if the ass stays at an empty plate then food may appear at that plate according to some fixed stochastic rule and if the ass stays at a plate (either full or empty) then the state of the other plate can change according to some other fixed (stochastic) rule. If the ass is in the middle then none of the plates can become empty in the next step (the ass is guarding the food). The dynamics can be summarized by a random mapping $t : X \times A \rightarrow X$ (or, equivalently, as a set of transition probabilities). The ass is considered to be consuming a unit food if he chooses to stay at a full plate. If x_t is the state at time t then the reinforcement streams $\{R_t, S_t\}$ of Eq. (1) can be given by $R_t = 1$ if in state x_t the ass is at a full plate and the chosen action, a_t , is "stay". $R_t = 0$, otherwise. Therefore, $R_t = R(x_t, a_t)$ for some function R . Further, $S_t = 1$, if the food disappears from a plate while the ass is at the other plate, otherwise $S_t = 0$. That is, $S_t = S(x_t, a_t, x_{t+1})$, where $x_{t+1} = t(x_t, a_t)$. Let us define the evaluation of a (deterministic, stationary) policy, $\pi : X \rightarrow A$, by

$$v_{\pi,1}(x) = \min\left(R_{\text{crit}}, E\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid x_0 = x\right]\right),$$

$$v_{\pi,2}(x) = E\left[\sum_{t=0}^{\infty} \gamma^t S_t \mid x_0 = x\right]$$

where $E[\cdot]$ is the expectation operator underlying the decision process. By standard arguments, and since $\min(R, E[\xi + \eta]) = \min(R, E[\xi] + E[\eta]) = \min(R, E[\xi] + \min(R, E[\eta]))$ holds if $R > 0$ and ξ, η are nonnegative random variables, one can show that v_{π} can be written recursively:

$$v_{\pi,1}(x) = \min\left(R_{\text{crit}}, R(x, \pi(x)) + \min\left(R_{\text{crit}}, \gamma \sum_{y \in X} p(x, \pi(x), y) v_{\pi,1}(y)\right)\right),$$

³Here A^B denotes the set of functions mapping B into A .

$$v_{\pi,2}(x) = \sum_{y \in X} p(x, \pi(x), y) \{S(x, \pi(x), y) + \gamma v_{\pi,2}(y)\}. \quad (2)$$

Here $p(x, a, y) = P(y = t(x, a))$. Similar recursions hold for non-deterministic, Markovian, and even for non-Markovian policies [Szepesvári, 1998]. Now, if one defines Q by

$$\begin{aligned} (Qv)(x, a)_1 &= \min(R_{\text{crit}}, R(x, a) + \min(R_{\text{crit}}, \\ &\quad \gamma \sum_{y \in X} p(x, a, y) v_1(y))), \\ (Qv)(x, a)_2 &= \sum_{y \in X} p(x, a, y) \{S(x, a, y) + \gamma v_2(y)\} \end{aligned}$$

and $T_\pi : \mathcal{R} \rightarrow \mathcal{R}$ by $(T_\pi v)(x) = (Qv)(x, \pi(x))$, $x \in X$, then we see that v_π becomes the fixed point of T_π . Note that the definition of Q is obtained from (2) by systematically replacing $\pi(x)$ by a , and v_π by v , meaning that Q provides a concise summary of both the state- and reinforcement-dynamics of the decision process in an abstract form.

Policies are compared on the basis of their evaluations. Since now $v_\pi(x) \in \mathcal{R} = \mathbf{R}^2$ is vector-valued we need a way to compare pairs of vectors. Therefore, we will assume that a binary relation \leq over \mathcal{R} is given which is reflexive, transitive and trichotomous (i.e., \leq is an *ordering*, or $\mathcal{R} = (\mathcal{R}; \leq)$ is a *lattice*).⁴ Buridan's ass requires a "reverse-2nd" *lexicographic ordering*: $r \leq r'$ if $r_1 < r'_1$ or if $r_1 = r'_1$ and $r_2 \geq r'_2$ (here the components of r and r' were denoted by lower indices). This finishes the construction of the ADP describing the problem-structure of Buridan's ass. This "reverse-2nd" lexicographic ordering differs from lexicographic ordering only by the condition on the second components: we wrote $r_2 \geq r'_2$ instead of $r_2 \leq r'_2$. For convenience, we will continue with considering lexicographic ordering. Lexicographic ordering (and also "reverse-2nd" ordering) satisfies the above properties, i.e., it is an ordering.

In order to facilitate the connection with RL we will define the notion of optimal value function (instead of relying on Pareto-optimality), but first we need to assign a meaning to the supremum of subsets of \mathcal{R} : for $A \subset \mathcal{R}$, $a = \text{s.u.p. } A$ is a value such that for all $c \geq A$,

⁴A binary relation \leq over \mathcal{R} is called i) *reflexive* if $r \leq r$ for any $r \in \mathcal{R}$; ii) *transitive* if r, r' and r'' are such that $r \leq r'$ and $r' \leq r''$ then $r \leq r''$ ($r, r', r'' \in \mathcal{R}$); and iii) *trichotomous* if for any pairs $(r, r') \in \mathcal{R}$ either $r \leq r'$ or $r' \leq r$ (the ordering is *total*) and if both relations hold then $r = r'$.

also $c \geq a$ ($a \geq b$ is defined by $b \leq a$, and $a \geq A$ is defined as $a \geq a'$ for all $a' \in A$). The infimum of sets is defined analogously. The maximum of a set A is defined by

$$a = \text{m.a.x. } A, \quad \text{iff } a \in A \quad \text{and} \quad a \geq b, \forall b \in A, \quad (3)$$

the minimum could be defined analogously. A lattice $(\mathcal{R}; \leq)$ is said to be *complete* if for all bounded subsets A , both the infimum and the supremum of the set exist. Lexicographic ordering can be made complete if the set of reals \mathbf{R} is replaced by the set of extended reals, $\bar{\mathbf{R}} = \{-\infty, +\infty\} \cup \mathbf{R}$, which is understood with the natural topology. Then if the return space is $\mathcal{R} = \mathbf{R}^2$, the supremum a^* of a set $A \subset \mathbf{R}^2$ can be defined in the standard way as follows: $a_1^* = \sup\{a_1 : a = (a_1, a_2)^T \in A\}$ and $a_2^* = \inf\{a_{2n} : a_n = (a_{1n}, a_{2n})^T \in A \text{ s.t. } a_{1n} \rightarrow a_1^*\}$.

The ordering \leq of \mathcal{R} is extended to functions assuming values in \mathcal{R} in the usual way: for $v, w \in \mathcal{R}^Y$ we say that $v \leq w$ iff for all $y \in Y$, $v(y) \leq w(y)$ holds. Note that the induced ordering, \leq , is only a partial ordering over \mathcal{R}^Y (i.e., it is not total).

Equipped with the notion of supremum we can define the optimal reinforcement function:

$$v^*(x) = \text{s.u.p.}_{\pi \in \Pi} v_\pi(x), \quad x \in X. \quad (4)$$

Here Π denotes a fixed set of policies. We will consider the case when Π equals to the set of all stationary policies. A policy in the class Π is said to be *optimal* if $v_\pi = v^*$.

Now, we can answer the question about the form of optimal stationary policies in the case of Buridan's ass. For sure, an "optimal ass" would indefinitely repeat "guarding steps" (staying in the middle) and "consumation steps". It should also be clear then that the exact ratio of the waiting periods would depend on the value of R_{crit} . It should also be clear that for some values of R_{crit} all stationary policies would be suboptimal. A form of optimal policies for this class of problems can be found in [Feinberg and Schwartz, 1995]. Note that if one extends the state space, so that the ass has counting-actions with a limited set of numbers (i.e. if the ass is enabled to count up to a fixed maximum number of steps) and if the ass can choose actions randomly then optimal policies w.r.t. the falls set of policies can be recovered exactly. So this case reduces to the case of randomized stationary policies. The following theorem restricts the set of policies further to deterministic stationary policies,

so that tractability of the *learning* problem will be ensured, but global optimality may be lost. The theorem is proven in the appendix.

Theorem 2.1 Consider a finite ADP, $(\mathcal{R}, X, A, \mathcal{A}, \mathcal{Q})$,⁵ where (i) $(\mathcal{R}; +, \lambda \cdot, \|\cdot\|_{\mathcal{R}})$ is a Banach-space and \mathcal{R} is equipped with (ii) a complete ordering \leq which satisfies the following countable transitivity property: (iii) if r_n is weakly convergent⁶ in \mathcal{R} , and $r_0 \leq r_1 \leq r_2 \leq \dots \leq r_n \leq r_{n+1} \leq \dots$ then $r_0 \leq \lim_{n \rightarrow \infty} r_n$. Further, assume that (iv) $\mathcal{Q} : \mathcal{R}^X \rightarrow \mathcal{R}^{X \times A}$ is monotone: $\mathcal{Q}v \leq \mathcal{Q}w$ whenever $v \leq w$, $v, w \in \mathcal{R}^X$, continuous in the topologies induced by pointwise convergence over \mathcal{R}^X and $\mathcal{R}^{X \times A}$, (v) and that \mathcal{Q} is a contraction w.r.t. the induced max-norm⁷ $\|\cdot\|_{\infty, \mathcal{R}}$. (vi) Assume that $T : \mathcal{R} \rightarrow \mathcal{R}$, defined by

$$(Tv)(x) = \max_{a \in A(x)} (\mathcal{Q}v)(x, a) \quad (5)$$

has a unique fixed point v^+ , and $\lim_{n \rightarrow \infty} T^n v = v^+$ for all $v \in \mathcal{R}^X$ s.t. $\|v\|_{\infty, \mathcal{R}} < \infty$. Let $\Pi = A^X$ be the space of stationary policies. Then (a) $v^+ \geq v_\pi$ for all π (π is a deterministic stationary policy) and $v^+ = v^*$, so $Tv^* = v^*$ (Bellman optimality equation); (b) if $T_\pi v^+ = Tv^+$, i.e., if π is myopic w.r.t. v^+ , then $v_\pi = v^*$ (myopic policies are optimal); (c) if $T_{\pi'} v_\pi > v_\pi$ then $v_{\pi'} \geq v_\pi$ (Howard's policy improvement routine is valid).

Operator T , as defined by (5), is called the *optimal value operator*.

It is easy to check that countable transitivity holds for sequences of \mathbf{R}^n and the lexicographic ordering. Note that contraction arguments cannot be used since there is no norm over \mathbf{R}^n with the lexicographic ordering for which the m.a.x. operator would be a non-expansion. For a further discussion of this and additional peculiarities related to lexicographic orderings see [Gábor et al., 1998].

Note that if $\mathcal{R} = \mathbf{R}^n$ with the lexicographic ordering then the actions at which the maximum is reached in

⁵An ADP $(\mathcal{R}, X, A, \mathcal{A}, \mathcal{Q})$ is called finite if both X and A are finite. The finiteness assumption could be relaxed by some extra work.

⁶A sequence r_n is said to be weakly convergent in \mathcal{R} if it is convergent in the topology induced by the vector space structure of \mathcal{R} .

⁷The induced maximum-norm $\|\cdot\|_{\infty, \mathcal{R}}$ is defined by $\|v\|_{\infty, \mathcal{R}} = \sup_{z \in Z} \|v(z)\|_{\mathcal{R}}$.

Eq. (5) can be computed by first computing the sets $A_{i+1} = \{a \in A_i(x) \mid \max_{b \in A_i(x)} (\mathcal{Q}f)(x, b)_i = (\mathcal{Q}f)(x, a)_i\}$ (6)

recursively for $i = 0, 1, 2, \dots, n-1$, with $A_0 = A(x)$. For convenience, we will denote the action sets as defined above by $A_i(Q, x)$ when $\mathcal{Q}f$ is replaced by any function $Q \in \mathcal{R}(X \times A)$:

$$\begin{aligned} A_0(Q, x) &= A(x) \\ A_{i+1}(Q, x) &= \{a \in A_i(Q, x) \mid \\ &\quad \max_{b \in A_i(Q, x)} Q(x, b)_i = Q(x, a)_i\}, \end{aligned}$$

where $i = 0, 1, 2, \dots, n-1$. Then $(Tv)(x)_{i+1} = \max_{a \in A_i(Qv, x)} (Qv)(x, a)_{i+1}$. Now we show that T has a unique fixed point and $T^n v$ converges to this fixed point for all bounded $v \in \mathcal{R}^X$ provided that \mathcal{Q} satisfies the conditions of the above theorem and if \mathcal{Q} acts *componentwise*, i.e., if $(Qv)_i = (Qw)_i$ whenever $v_i = w_i$:

Theorem 2.2 Assume that \mathcal{Q} acts componentwise and that conditions (i)–(v) of Theorem 2.1 are satisfied. Then also condition (vi) is satisfied and thus the conclusions of Theorem 2.1 hold.

Proof. Fix v and consider the first component of $T^n v$. Define $T_1 : \mathbf{R}^X \rightarrow \mathbf{R}^X$ by $T_1 f = (T\hat{f})_1$, where $\hat{f} = (f, f_2, \dots, f_n)$ with f_2, \dots, f_n being arbitrary. T_1 is well defined and is a contraction. Moreover, $(T^n v)_1 = T_1^n v_1$ holds for all $n \in \mathbf{N}$, and therefore $(T^n v)_1$ converges to the unique fixed point of T_1 . Similarly, if u and w are both fixed points of T then $u_1 = w_1$. Let us denote this common value by v_1^+ . Now, consider $(T^n v)_2$. Since $(T^{n+1} v)_2(x) = \max_{a \in A_1(QT^n v, x)} (QT^n v)(x, a)_2$, and since \mathcal{Q} is componentwise, $A_1(QT^n v, x)$ depends only on $(T^n v)_1$ which is known to converge. Therefore, because of the finiteness of A , for n large enough $A_1(QT^n v, x)$ will stabilize at some set $A_1^*(v, x)$. Now, since the operator $u(x) \mapsto \max_{a \in A_1^*(v, x)} (\mathcal{Q}\hat{u})(x, a)_2$ is a contraction, where $\hat{u} = (v_1^+, u, u', \dots)$, also $(T^n v)_2$ will converge to some value (the operator is well defined since \mathcal{Q} is componentwise). Moreover, if u and w are both fixed points of T then $u_1 = w_1$ and thus $A_1(Qu, x) = A_1(Qw, x) (= A_1^*(x))$ for all $x \in X$, and so u_2 and w_2 are both the fixed points of the contraction $z \mapsto \max_{a \in A_1^*(x)} (\mathcal{Q}\hat{z})(x, a)_2$ and are therefore equal. Continuing in this way for the higher indices we get the proof of the required statement. Q.u.e.d.

The above theorem shows that the dilemma of Buridan's ass is indeed in the realm of Theorem 2.1, since

the appropriate reinforcement propagator operator, Q , acts componentwise.

Theorem 2.1 is just one example of how the existence of optimal stationary policies can be ensured in multi-criteria problems. There are many possible extensions of it, but these are outside of the scope of the present article.

3 Learning optimal policies

Since most convergence proofs for RL algorithms rely on contraction arguments the generalization of results like the convergence of such as the Adaptive Real-Time Dynamic Programming [Barto et al., 1991], Q-learning [Watkins, 1990], TD(λ) [Sutton, 1988] are easy to obtain for vector-valued MDPs *provided* that T is a *contraction*⁸. Unfortunately, this will hold rarely. Nevertheless a componentwise analysis, similar to the one presented at the end of the previous section, will in general yield the desired convergence result.

As a particular example consider the case of Q-learning. Let $Q^* = Qv^*$ be the optimal action-value function. Q-learning solves the fixed point equation $Q^* = QSQ^*$, $(SQ)(x) = \max_{b \in A(x)} Q(x, b)$, by relaxation and without ever estimating Q . In the case of an MDP with the expected discounted total cost Q-learning takes the form

$$Q_{t+1}(x_t, a_t) = (1 - \alpha_t(x_t, a_t))Q_t(x_t, a_t) + \alpha_t(x_t, a_t) \{ R_t(x_t, a_t, x_{t+1}) + \gamma \max_{b \in A(x_t)} Q_t(x_{t+1}, b) \},$$

with $Q_{t+1}(x, a) = Q_t(x, a)$ for pairs $(x, a) \neq (x_t, a_t)$. The relaxation factor (learning rate) $0 < \alpha_t(x_t, a_t) < 1$ is gradually decreased towards zero so that the variance of the estimates are reduced and (probability one) convergence can be achieved.

A raw generalization of Q-learning to vector-valued Q-learning would replace the immediate-reward scalars (R_t) in the above equation by immediate-reward vectors and “max” by “m.a.x.” (remember that m.a.x. is the maximum element of A according to the chosen ordering \leq of \mathcal{R} – see Eq. (3) for the definition of m.a.x.). For simplicity, consider a two-dimensional return space with the lexicographic ordering and a componentwise reinforcement propagation scenario when

⁸In fact, since the convergence of the vast majority of RL algorithms follows from the general asynchronous contraction-mapping theorem of [Littman and Szepesvári, 1996] (see also [Szepesvári and Littman, 1997]), it is sufficient to reproduce the proof of that theorem. It turns out, that the raw generalization of that proof will work without any problems *for contractions*. However, this is out side of the scope of this article.

the components are computed by some expected value criteria. Proceeding componentwise, we see that the update equation for the first component is left intact, but the update of the second component becomes

$$Q_{t+1,2}(x_t, a_t) = (1 - \alpha_t(x_t, a_t))Q_{t,2}(x_t, a_t) + \alpha_t(x_t, a_t) \{ R_{t,2}(x_t, a_t, x_{t+1}) + \gamma \max_{b \in A_1(Q_t, x_t)} Q_{t,2}(x_{t+1}, b) \},$$

where $A_i(Q, x)$ is defined by Eq. (6). Note that in the example of Buridan’s ass the first criterion is a truncated evaluation criterion and thus must be treated differently. Unfortunately, due to the lack of space we cannot present the direct learning rules for this criterion, but we note here that this rule can be obtained almost entirely automatically if one tries to estimate $Z^*(x, a) = \sum_y p(x, a, y)Q_1^*(x, a)$ instead of Q_1^* [Szepesvári and Littman, 1997]. Note that knowing Z^* alone is insufficient to recover Q_1^* . Therefore either one estimates $R(x, a)$ and then computes $Q_1^*(x, a)$ or, one may estimate Q_1^* in a second update rule using the estimates of Z^* directly, without ever estimating $R(x, a)$. This latter rule will work only if $R(x, a)$ is deterministic. The convergence of these algorithms follows by the standard proofs completed with a componentwise analysis.

The analogue of Q-learning for MDPs with the maximin criterion, proposed by Heger [Heger, 1994, 1996], is the Q-hat algorithm given by

$$Q_{t+1}(x_t, a_t) = \min \{ Q_t(x_t, a_t), R_t(x_t, a_t, x_{t+1}) + \gamma \max_{b \in A} Q_t(x_{t+1}, b) \}.$$

This algorithm will converge to the optimal Q-function if $Q_0 \geq Q^*$ (the initial estimate is optimistic). The raw generalization replaces “min” and “max” by “m.i.n.” and “m.a.x.”, respectively. Unfortunately, this generalization may fail to converge to Q^* since the convergence of Q-hat exploits $Q_t \geq Q^*$ ($t \geq 0$) and this may become invalid in this case.⁹ In order to surmount this problem one has to update the second and larger index components by some means other than Q-hat learning.

It is natural then to consider adaptive real-time dynamic programming algorithms. For maximin prob-

⁹This can be shown in the following way: Consider again $\mathcal{R} = \mathbf{R}^2$ with the lexicographic ordering. Then $Q_{t+1,2}(x_t, a_t) = \min \{ Q_{t,2}(x_t, a_t), R_{t,2}(x_t, a_t, x_{t+1}) + \gamma \max_{b \in A_1} Q_{t,2}(x_{t+1}, b) \}$, where $A_1 = A_1(Q_t, x_t)$. Notice that $Q_{t+1,2}(x, a) \leq Q_{t,2}(x, a)$ for all $(x, a) \in U$ so if once $Q_{t,2}(x, a) < Q_2^*(x, a)$ then $Q_{t,2}(x, a)$ cannot converge to $Q_2^*(x, a)$. Here, $A_1(Q_t, x_t)$ may be quite different from $A_1(Q^*, x_t)$ which means that $Q_{t+1,2}(x_t, a_t)$ may become smaller than $Q^*(x_t, a_t)$ even if $Q_{t,2} = Q_2^*$, depending only on the values of $Q_{t,1}$.

lems this algorithm builds an estimate of the transition sets $T(x, a) = \{y \in X \mid p(x, a, y) > 0\}$ using

$$T_{t+1}(x_t, a_t) = T_t(x_t, a_t) \cup \{x_{t+1}\}$$

and another estimate of the rewards $R(x, a, y)$ by $R_{t+1}(x_t, a_t, x_{t+1}) = R_t$, where x_t, a_t are the state and action at time t , and where $R_t \in \mathbf{R}^2$ is the immediate reward vector at time t . The value function estimate $v_t(x) \in \mathbf{R}^2$ is updated by the equation

$$v_{t+1}(x_t) = \max_a \min_{y \in T_{t+1}(x_t, a_t)} (R_t(x_t, a_t, y) + \gamma v_t(y)).$$

Since there is no “optimistic initialization” condition here, one may show (using a componentwise analysis) that this algorithm converges to optimality if some other conditions ensuring “sufficient exploration” hold. Further discussions related to action-selection strategies ensuring “sufficient exploration” in minimax problems can be found in [Gábor et al., 1998].

4 Computer simulations

The purpose of the computer simulations was twofold: to demonstrate that the theory works in practice, and to provide some hints on the rate of convergence of different algorithms. The ARTDP algorithm were tried out for tic-tac-toe with lexicographic ordering. The first criterion prescribed the desire to win (or make a draw) and the second to finish the game as soon as possible.¹⁰ The action selection procedure was the greedy policy in all of the cases, i.e., $Q_t(x, a(t)) = \max_a Q_t(x, a)$ for each t . Several opponents were tried whose strategy was a mixture of the optimal minimax policy (computed by $\alpha - \beta$ -pruning with ties broken randomly) and a totally randomized one. The degree of randomness was set to 0, 0.25, 0.5, 0.75 and 1, so that the first opponent, corresponding to randomness 0, is the optimal one, while the last one is the totally randomized one. For comparison both the multi-criteria and single criterion ARTDP algorithms were tried (called MC-ARTDP and ARTDP, respectively.) The learner started the game in each trial. The percent of wins and draws, and the number of steps in the cases of won or drew games are shown in

¹⁰The first component of the reinforcement-vector was +1 if the learner won, 0 if the game was a draw and -1 if he lost the game. The second component was unity in each step. We used the well known minimax representation of alternating games [see e.g. Littman and Szepesvári, 1996]. Note that by a simple change to the lexicographic ordering one may consider another criterion when the learner minimizes the number of steps only when starting from winning states, otherwise trying to mark time.

		0	0.25	0.5	0.75	1
ARTDP	Win or draw	0.73	0.74	0.74	0.76	0.74
	Steps	3.55	4.2	4.18	4.18	4.19
MC-ARTDP	Win or draw	0.85	1	0.96	1	1
	Steps	3.59	3.28	3.29	3.28	3.28

Table 1: Results of exhaustive testing. Percents of optimal moves learnt, and average number of steps to the end of the game for cases when the learner won are shown for both learners learning with ARTDP and MC-ARTDP. In the first row the degree of randomness of the opponents are shown: a randomness of 0 means an optimal opponent, while a randomness of 1 means a perfectly random opponent. The results suggest that since the learners do not explore, a complete optimal policy cannot be learned against the perfect opponent (just part of the game-tree is explored). The number of steps until the end of the game are consistently smaller for MC-ARTDP than that of for ARTDP. Also MC-ARTDP can win a larger percent of games.

Table 1. The percents are computed by employing an exhaustive search, i.e., the percent of those leaves in the *full reachable* game-tree when our learner did not lose the game was measured. As expected, the number of steps until the end of game is lower on average for the MC-ARTDP algorithm than that of for the ARTDP algorithm. Note that this comparison is not entirely satisfactory since this number is computed just for a part of the games and this obviously distorts the results. The effect of this can be observed in the statistics of the games played against the perfect opponent: apparently here MC-ARTDP needed more steps than ARTDP, but since MC-ARTDP won a larger percent of games this increase can be accounted for the games that MC-ARTDP won (or drew) and ARTDP lost. Intriguingly, the results also show that MC-ARTDP performs better than ARTDP in all of the cases, i.e., it could explore a larger part of the game-tree. We conjectured that the reason for this is that MC-ARTDP uses more information than ARTDP. In particular, since the second components of its evaluation function are initialized to zero, initially unexplored actions will look more favourable than explored ones, meaning that dependence on the second component will facilitate exploration. To confirm the conjecture we ran another set of experiments using the ARTDP algorithm and when actions were chosen based on one of the following two well-known exploration strategies: the Boltzmann-exploration and the ϵ -greedy strategy with decaying

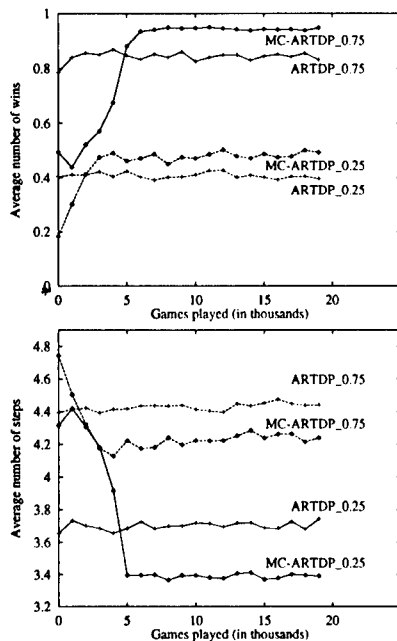


Figure 1: Results of learning with the one-criterion and multi-criteria ARTDP algorithms against opponents of different strengths. MC-ARTDP-0.25 and MC-ARTDP-0.75 label the curves of MC-ARTDP for an opponent with randomness 0.25 and 0.75, respectively.

exploration.¹¹ In this case ARTDP yielded comparable results to that of MC-ARTDP, thus confirming the hypothesis.¹²

Exploration has a price, though. The more exploratory actions the player tries the larger is the number of games lost during the learning trials. In order to get a more complete picture about the performances of the two algorithms we have measured on-line (or during-learning) performance. Results are shown in Figures 1. The upper subfigure shows the percent of plays won or drew. The larger the convergence speed to 1 is, the smaller is the cost of exploration. The lower subfigure depicts the number of steps until the end of the game, for the games when our learner actually won. Both figures show results for the opponents with randomness 0.25 and 0.75 (results for the other cases can

¹¹The ϵ -greedy exploration strategy chooses the best-looking (greedy) action with probability $1 - \epsilon$ and chooses an action uniformly randomly from the rest with probability ϵ [Thrun, 1992].

¹²In theory, as time goes to infinity both algorithms will converge to optimality. So the worse than optimal results should not be considered as cases when the algorithms got stuck in "local minima".

be roughly obtained by intra- and extrapolations and are not shown). Note that both the ARTDP and MC-ARTDP learn faster against weaker opponents which could be accounted for the small average depth of visited game tree when playing against a weak opponent. Note that the learner trained against a weak opponent will probably fail to win over a strong one, and the reverse may hold, too: in order to learn the optimal minimax strategy the opponents should not be restricted¹³. Also, in the case of both opponents MC-ARTDP learns slightly slower (in the short-term) but results in a better policy in the medium-term. More experiments are needed to analyze these findings.

5 Conclusions

We have considered multi-criteria decision problems in the framework of abstract dynamic programming. The reinforcements were assumed to be vector-valued and were compared by a total ordering defined over an appropriate vector space. A result, showing the existence of optimal policies was derived and it was shown that it applies to lexicographic ordering when the reinforcement propagation works "componentwise". Next, reinforcement learning algorithms were derived for this case and we have argued that their convergence can be proven by componentwise analysis. Experimental results were presented to illustrate the behavior of the algorithms. In the future we plan to extend the results and run other simulations to reinforce the utility of multi-criteria learning.

Acknowledgements

This work was partially done while Cs.Sz. was with the Research Group of Artificial Intelligence. This work was partially supported by OTKA Grant No. F20132 and the Hungarian Ministry of Education Grant No. FKFP 1354/1997.

Appendix

Here we prove Theorem 2.1, the text of which is not repeated here because of lack of space. Firstly, we

¹³Since the opponents are randomized (except the optimal opponent) the algorithms will eventually converge to optimality. However, the convergence rate will still depend on the degree of randomness of the opponent. The convergence rate will depend on how fast can the part of the game-tree which is accessible for an optimal player be fully explored. For opponents with higher randomness deep parts can hardly be accessed, for opponents with small randomness parts that follow an initial sub-optimal choice will be hard to explore.

shall prove that v^+ , the unique fixed point of T , majorizes the optimal value function, v^* . Fix an arbitrary policy π and observe that $Tv_\pi \geq T_\pi v_\pi$. Since $T_\pi v_\pi = v_\pi$, also $Tv_\pi \geq v_\pi$. From this, and because of the monotonicity of T (which holds because A is finite), we obtain $T^2v_\pi \geq Tv_\pi \geq v_\pi$. Iterating this indefinitely, we get that $T^{n+1}v_\pi \geq T^n v_\pi \geq \dots \geq v_\pi$ holds for all $n \in \mathbb{N}$. Thus, $T^n v_\pi$ is monoton increasing and thus (by the countable transitivity assumption) $\lim_{n \rightarrow \infty} T^n v_\pi \geq v_\pi$. Now, since $\lim_{n \rightarrow \infty} T^n v_\pi = v^+$, so $v^+ \geq v_\pi$. Since π was arbitrary, it follows that $v^+ \geq v^*$ by the definition of the s.u.p. operator. Now, let π be a policy which is myopic w.r.t. v^+ : $T_\pi v^+ = Tv^+$. Since $Tv^+ = v^+$, so $T_\pi v^+ = v^+$. Now, since v_π is the unique fixed point of T_π (T_π is a contraction since Q is a contraction), we get that $v^+ = v_\pi$. This shows that $v^+ = v^*$ and that π is optimal. In order to prove the third part consider a pair of policies (π, π') s.t. $T_{\pi'} v_\pi > v_\pi$. By the first train of thoughts, we get that $T_{\pi'}^n v_\pi \geq v_\pi$ is a monotone increasing sequence, so that $v_{\pi'} = \lim_{n \rightarrow \infty} T_{\pi'}^n v_\pi \geq v_\pi$ holds, too, thus finishing the proof.

References

- E. Altman and A. Schwartz. Adaptive control of constrained Markov chains: Criteria and policies. *Annals of Operations Research*, 28:101–134, 1991.
- M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. Coordination of multiple behaviors acquired by a vision-based reinforcement learning. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robot and Systems*, volume 2, pages 917–924, 1994.
- A.G. Barto, S.J. Bradtke, and S.P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical report 91-57, Computer Science Department, University of Massachusetts, 1991.
- D. P. Bertsekas. Monotone mappings with application in dynamic programming. *SIAM J. Control and Optimization*, 15(3):438–464, 1977.
- T.A. Brown and R.E. Strauch. Dynamic programming on multiplicative lattices. *J. Math. Anal. and App.*, 12:364–370, 1965.
- E.V. Denardo. Contraction mappings in the theory underlying dynamic programming. *SIAM Rev.*, 9:165–177, 1967.
- E.A. Feinberg. Controlled Markov decision process with arbitrary numerical criteria. *Theory of Probability and Applications*, 27:486–503, 1982.
- E.A. Feinberg and A. Schwartz. Markov decision models with weighted discounted rewards. *Mathematics of Operations Research*, 19:152–168, 1994.
- E.A. Feinberg and A. Schwartz. Constrained Markov decision models with weighted discounted rewards. *Mathematics of Operations Research*, 20(2):302–320, 1995.
- E.B. Frid. On optimal strategies in control problems with constraints. *Theory of Probability and Applications*, 17: 188–192, 1972.
- Z. Gábor, Zs. Kalmár, and Cs. Szepesvári. Multi-criteria reinforcement learning. Technical report 98-115, Research Group on Artificial Intelligence, JATE-MTA, 1998.
- M. Heger. Consideration of risk in reinforcement learning. Revised submission to the 11th International Machine Learning Conference ML-94, 1994.
- M. Heger. The loss from imperfect value functions in expectation-based and minimax-based tasks. *Machine Learning*, 22:197–225, 1996.
- M.I. Henig. Vector-valued dynamic programming. *SIAM J. Control and Optimization*, 21(3):490–499, 1983.
- D. Heyman and M. Sobel. *Stochastic Models in Operations Research: Stochastic Optimization*, volume 2. McGraw-Hill, New York, 1984.
- M.L. Littman and Cs. Szepesvári. A Generalized Reinforcement Learning Model: Convergence and applications. In *Int. Conf. on Machine Learning*, pages 310–318, 1996.
- L.G. Mitten. Composition principles for synthesis of optimum multi-stage processes. *Operations Research*, 12: 610–619, 1964.
- S.M. Ross. *Applied Probability Models with Optimization Applications*. Holden Day, San Francisco, California, 1970.
- S. Singh and D. Cohn. How to dynamically merge Markov decision processes. In *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1997. MIT Press. in press.
- M.J. Sobel. Ordinal dynamic programming. *Management Science*, 21:967–975, 1975.
- R.S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
- Cs. Szepesvári. Non-markovian policies in sequential decision problems. *Acta Cybernetica*, 1998. (accepted).
- Cs. Szepesvári and M.L. Littman. A unified analysis of value-function-based reinforcement-learning algorithms. 1997. (submitted).
- S.B. Thrun. *The role of exploration in learning control*. Van Nostrand Reinhold, Florence KY, 1992.
- C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1990.

Local Cascade Generalization

João Gama

LIACC, FEP - University of Porto

Rua Campo Alegre, 823

4150 Porto, Portugal

Phone: (+351) 2 6078830 Fax: (+351) 2 6003654

Email: jgama@ncc.up.pt

<http://www.up.pt/liacc/ML>

Abstract

In a previous work we have presented *Cascade Generalization*, a new general method for merging classifiers. The basic idea of Cascade Generalization is to sequentially run the set of classifiers, at each step performing an extension of the original data by the insertion of new attributes. The new attributes are derived from the probability class distribution given by a base classifier. This constructive step extends the representational language for the high level classifiers, relaxing their bias. In this paper we extend this work by applying Cascade *locally*. At each iteration of a *divide and conquer* algorithm, a reconstruction of the instance space occurs by the addition of new attributes. Each new attribute represents the probability that an example belongs to a class given by a base classifier. We have implemented three *Local Generalization Algorithms*. The first merges a linear discriminant with a decision tree, the second merges a naive Bayes with a decision tree, and the third merges a linear discriminant and a naive Bayes with a decision tree. All the algorithms show an increase of performance, when compared with the corresponding single models. *Cascade* also outperforms other methods for combining classifiers, like *Stacked Generalization* and competes well against *Boosting*, with statistically significant confidence levels.

Keywords: Multiple Models, Constructive Induction, Merging Classifiers.

1 Introduction

The ability of a chosen algorithm to induce a good generalization depends on how appropriate the class model underlying the algorithm is for the given task. An algorithm class model is the representation language it uses to express a generalization of the examples. The representation language for a standard decision tree is the DNF formalism that splits the instance space by axis-parallel hyper-planes, while the representation language for a linear discriminant function is a set of linear functions that split the instance space by oblique hyper-planes. Since different learning algorithms employ different knowledge representations and search heuristics, different search spaces are explored and diverse results are obtained. The problem of finding the appropriate bias for a given task is an active research area. We can consider two main lines: on one hand methods that try to select the most appropriate algorithm for the given task, for instance Schaffer's selection by Cross-Validation, and on the other hand, methods that combine predictions of different algorithms, for instance Stacked Generalization [25]. This work follows the second research line. Instead of looking for methods that fit the data using a single representation language, we present a family of algorithms, under the generic name of *Cascade Generalization*, whose search space contains models that use different representation languages. Cascade generalization was first presented in [14]. It performs an iterative composition of classifiers. At each iteration a classifier is generated. The input space is extended by the addition of new attributes. These are in the form of a probability class distribution which are obtained, for each example, by the generated base classifier. The language of the final classifier is the language used by the high level generalizer. This language uses terms that are expressions from the language of low

level classifiers. In this sense, Cascade Generalization generates a unified theory from the base theories.

Here we extend the work presented in [14], by applying *Cascade* locally. In our implementation, *Local Cascade Generalization* generates a decision tree. The experimental study shows that this methodology usually improves both accuracy and theory size with statistical significance levels.

The next section presents the framework of *Cascade Generalization*. In section 3 we define a new family of algorithms that apply *Cascade Generalization locally*. In section 4 we review previous work in the area of multiple models. In section 5, we perform an empirical study using UCI data sets. The last section presents an analysis of the results and concludes the paper.

2 Cascade Generalization

Consider a learning set $D = (\vec{x}_n, y_n) \ n = 1, \dots, N$, where $\vec{x}_n = [x_1, \dots, x_m]$ is a multidimensional input vector, and y_n is the output variable. Since the focus of this paper is on classification problems, y_n takes values from a set of predefined values, that is $y_n \in \{Cl_1, \dots, Cl_c\}$, where c is the number of classes. A classifier \mathfrak{S} is a function that is applied to the training set D in order to construct a model $\mathfrak{S}(D)$. The generated model is a mapping from the input space X to the discrete output variable Y . When used as a predictor, represented by $\mathfrak{S}(\vec{x}, D)$, it assigns a y value to the example \vec{x} . This is the traditional framework for classification tasks. Our framework requires that the predictor $\mathfrak{S}(\vec{x}, D)$ outputs a vector representing conditional probability distribution $[p_1, \dots, p_c]$, where p_i represents the probability that the example \vec{x} belongs to class i , i.e. $P(y = Cl_i | \vec{x})$. The class that is assigned to the example \vec{x} , is the one that maximizes this last expression. Most of the commonly used classifiers, such as *naive Bayes* and *Discriminant*, classify each example in this way. Other classifiers, for example *C4.5*, have a different strategy for classifying an example, but it requires small changes to obtain a probability class distribution.

We define a constructive operator $\Phi(D', \mathfrak{S}(\vec{x}, D))$. This operator has two input parameters: a data set D' and a predictor $\mathfrak{S}(\vec{x}, D)$. The classifier \mathfrak{S} generates a theory from the training data D . For each example $\vec{x} \in D'$, the generated theory outputs a probability class distribution. For all the examples in D' the operator Φ concatenates the input vector \vec{x} with the output probability class distribution. The output of $\Phi(D', \mathfrak{S}(\vec{x}, D))$ is a new data set D'' . The cardinal-

ity of D'' is equal to the cardinality of D' (i.e. they have the same number of examples). Each example in $\vec{x} \in D''$ has an equivalent example in D' , but augmented with c new attributes. The new attributes are the elements of the vector of class probability distribution obtained when applying classifier $\mathfrak{S}(\vec{x}, D)$ to the example \vec{x} . Cascade generalization is a sequential composition of classifiers, that at each generalization level applies the Φ operator. Given a training set L , a test set T , and two classifiers \mathfrak{S}_1 , and \mathfrak{S}_2 , Cascade generalization proceeds as follows:

Using classifier \mathfrak{S}_1 , generates the *Level₁* data:

$$\begin{aligned} Level_1 train &= \Phi(L, \mathfrak{S}_1(\vec{x}, L)) \\ Level_1 test &= \Phi(T, \mathfrak{S}_1(\vec{x}, L)) \end{aligned}$$

Classifier \mathfrak{S}_2 learns on *Level₁* training data and classifies the *Level₁* test data:

$$\mathfrak{S}_2(\vec{x}, Level_1 train) \text{ for each } \vec{x} \in Level_1 test$$

Those steps perform the basic sequence of a cascade generalization of classifier \mathfrak{S}_2 after classifier \mathfrak{S}_1 . We represent the basic sequence by the symbol ∇ .

The previous composition could be shortly represented by:

$$\mathfrak{S}_2 \nabla \mathfrak{S}_1 = \mathfrak{S}_2(\vec{x}, Level_1 train) \text{ for each } \vec{x} \in Level_1 test$$

which is equivalent to:

$$\mathfrak{S}_2 \nabla \mathfrak{S}_1 = \mathfrak{S}_2(\vec{x}, \Phi(L, \mathfrak{S}_1(\vec{x}', L))) \text{ for each } \vec{x} \in \Phi(T, \mathfrak{S}_1(\vec{x}'', L))$$

This is the simplest formulation of *Cascade Generalization*. Some possible extensions include the composition of n classifiers, and the parallel composition of classifiers.

A composition of n classifiers is represented by:

$$\mathfrak{S}_n \nabla \mathfrak{S}_{n-1} \nabla \mathfrak{S}_{n-2} \dots \nabla \mathfrak{S}_1$$

In this case, Cascade Generalization generates $n-1$ levels of data. The high level theory, is that one given by the \mathfrak{S}_n classifier.

A variant of cascade generalization, which includes several algorithms in parallel, could be represented in this formalism by:

$$\begin{aligned} \mathfrak{S}_{n+1} \nabla [\mathfrak{S}_1, \dots, \mathfrak{S}_n] &= \\ \mathfrak{S}_{n+1}(\vec{x}, \Phi(L, [\mathfrak{S}_1(\vec{x}', L), \dots, \mathfrak{S}_n(\vec{x}', L)])) & \\ \text{for each } \vec{x} \in \Phi(T, [\mathfrak{S}_1(\vec{x}'', L), \dots, \mathfrak{S}_n(\vec{x}'', L)]) & \end{aligned}$$

The algorithms $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ run in parallel. The operator $\Phi(L, [\mathfrak{S}_1(\vec{x}, L), \dots, \mathfrak{S}_n(\vec{x}, L)])$ returns a new data set L' which contains the same number of examples as L . Each example in L' contains $n * c$ new attributes, where c is the number of classes. Each algorithm in the set $\mathfrak{S}_1, \dots, \mathfrak{S}_n$ contributes with c new attributes.

3 Local Cascade Generalization

Most of Machine Learning algorithms for supervised learning use a divide and conquer strategy that attacks a complex problem by dividing it into simpler problems and recursively applies the same strategy to the subproblems. Solutions of sub-problems can be combined to yield a solution of the complex problem. This is the basic idea behind well known decision tree based algorithms: ID3 (Quinlan, 1984), ASSISTANT (Kononenko et al., 1987), CART (Breiman et al., 1984), C4.5 (Quinlan, 1993), etc. The power of this approach comes from the ability to split the hyperspace into subspaces and fit each subspace with different functions. In our previous work [14] we have shown that *Cascade* significantly improves the performance of this type of learning algorithms. In this paper we explore the applicability of *Cascade* on the problems and subproblems that a *divide and conquer* algorithm must solve. The intuition behind this hypothesis is the same as behind any *divide and conquer* strategy. The relations that can not be captured at global level can be discovered on the simpler subproblems.

Local cascade generalization, is a composition of algorithms that is performed for each task when building the classifier. At each iteration of a divide and conquer algorithm, local cascade generalization will be performed by applying the Φ operator. The effect is that the input space is reconstructed by the insertion of the new attributes. These new attributes are propagated down to the subtasks that the algorithm might consider. In this paper we restrict the use of *local Cascade Generalization* to decision tree based algorithms. However, it would be possible to use it with any *divide and conquer* algorithm. Figure 1 presents the general algorithm of *local Cascade Generalization*, applied to a decision tree.

When growing the tree, at each decision node new attributes are computed by applying the Φ operator. The new attributes that are created there are propagated down the tree. The number of new attributes is equal to the number of classes of the examples that fall at this node. At different levels, the algorithm considers data sets with different number of attributes and

Input: A data set D , a base classifier \mathfrak{S}

Output: A decision Tree

Function CGtree(D, \mathfrak{S})

IF stop criteria(D) = TRUE

return a Leaf with class probability distribution

$D' = \Phi(D, \mathfrak{S}(\vec{x}, D))$

 Choose the attribute that maximizes

 splitting criterion on D'

For each partition of examples based on

 chosen attribute values

$Tree_i = \text{CGtree}(D'_i, \mathfrak{S})$

return Tree as a decision node based on

 chosen attribute, storing $\mathfrak{S}(D)$

 and descendants $Tree_i$

End

Figure 1: Local Cascade Algorithm based on a Decision Tree

classes. Deeper nodes contain an increasing number of attributes. This could be a disadvantage of the system, but the number of new attributes is not constant. As the tree grows and the classes are discriminated, deeper nodes also contain examples from a decreasing number of classes. This means that as the tree grows the number of new attributes decreases.

In order to be applied as a predictor, any *CGTree* must store, at each node, the model generated by the base classifier using the examples that fall at this node. When classifying a new example, the example traverses the tree in the usual way, but at each decision node it is extended by the insertion of the probability class distribution provided the base classifier predictor at this node.

In the framework of local cascade generalization, we have developed a *CGLtree*, that uses the $\Phi(D, \text{Discrim}(\vec{x}, D))$ operator in the constructive step. Each internal node of a *CGLtree* contains a discriminant function. This discriminant function is used to build new attributes. For each example \vec{x} , the value of a new attribute A_i is computed using the probability $p(C_i|\vec{x})$ which is given by the linear discriminant function. At each decision node, the number of new attributes built by *CGLtree* is always equal to the number of classes taken from the examples that fall at this node. We use the following heuristic: we only consider a *class_i* if the number of examples, at this node, belonging to *class_i* is greater than N times the number of attributes¹. By default N is 3. This implies

¹This heuristic was suggested by Breiman et al.[3]

that at different nodes, different number of classes will be considered and a different number of new attributes is added.

In our empirical study we have used two other algorithms that locally apply *Cascade Generalization*. *CGBtree* that uses as constructive operator $\Phi(D, \text{naiveBayes}(\vec{x}, D))$, and *CGBLtree* that uses as constructive operator:

$\Phi(D, [\text{naiveBayes}(\vec{x}, D), \text{Discrim}(\vec{x}, D)])$. In all other aspects these algorithms are similar to *CGLtree*.

There is one restriction to the application of the $\Phi(D', \mathfrak{S}(x, D))$ operator: the \mathfrak{S} classifier must return a probability class distribution for each $x \in D'$. Any classifier that satisfies these requisites could be applied. It is possible to imagine a *CGTree*, whose internal nodes are trees themselves. For example, small modifications to C4.5², will allow the construction of a *CGTree* whose internal nodes are trees generated by C4.5.

4 Related Work

With respect to the final model, there are clear similarities between *CGLtree* and *Multivariate trees* [5, 15]. Any multivariate tree is topologically equivalent to a three-layer *inference network* [18]. The constructive ability of our system is similar to the *Cascade Correlation Learning architecture* [11]. Also the final model of *CGBtree* is related with the *recursive naive Bayes* presented in [17]. In a previous work [13], we have compared system *Ltree*, similar to *CGLtree*, with *Oc1* [19] and *LMDT* [5]. The focus of this paper is on methodologies for combining classifiers. As such, we review other methods that generate and combine multiple models.

4.1 Combining Classifications

We can consider two main lines of research. One group includes methods where all base classifiers are consulted in order to classify a query example. The other includes methods that characterize the area of expertise of the base classifiers and for a query point only ask the opinion of the experts. Voting is the most common method used to combine classifiers. As pointed out by Ali and Pazzani [1], this strategy is motivated by the Bayesian learning theory which stipulates that in order to maximize the predictive accuracy, instead of using just a single learning model, one should ideally use all models in the hypothesis space. The vote

of each hypothesis should be weighted by the posterior probability of that hypothesis given the training data. Several variants of the voting method can be found in the machine learning literature. From uniform voting where the opinion of all base classifiers contributes to the final classification with the same strength, to weighted voting, where each base classifier has a weight associated, that could change over the time, and strengthens the classification given by the classifier.

Ortega [20] presents the "*Model Applicability Induction*" approach for combining predictions from multiple models. The approach consists of learning for each available model a *referee* that characterize situations in which each of the models is able to make correct predictions. In future instances these referees are first consulted to select the most appropriate prediction model and the prediction of the selected model is then returned.

4.2 Generating different models

Several methods for generating multiple models appear in the literature. Breiman [3] proposes *bagging*, that produces replications of the training set by sampling with replacement. Each replication of the training set has the same size as the original data, but some examples do not appear in it, while others may appear more than once. From each replication of the training set a classifier is generated. All classifiers are used to classify each example in the test set, usually using a uniform vote scheme.

The boosting algorithm of Freund and Schapire [12] maintains a weight for each example in the training set that reflects its importance. Adjusting the weights causes the learner to focus on different examples leading to different classifiers. Boosting is an iterative algorithm. At each iteration the weights are adjusted in order to reflect the performance of the corresponding classifier. The weight of the misclassified examples is increased. The final classifier aggregates the learned classifiers at each iteration by weighted voting. The weight of each classifier is a function of its accuracy.

Wolpert [25] proposed Stacked Generalization, a technique that uses learning in two levels. A learning algorithm is used to determine how the outputs of the base classifiers should be combined. The original data set constitutes the level zero data. All the base classifiers run at this level. The level one data are the outputs of the base classifiers. Another learning process occurs using as input the level one data and as output the

²Two different methods are presented in [14, 23].

final classification. This is a more sophisticated technique of cross validation that could reduce the error due to the bias.

Brodley [4] presents *MCS*, a hybrid algorithm that combines, in a single tree, nodes that are *univariate tests*, *multivariate tests* generated by *linear machines* and *instance based learners*. At each node *MCS* uses a set of *If-Then* rules to perform a hill-climbing search for the best hypothesis space and search bias for the given partition of the dataset. The set of rules incorporates knowledge of experts. *MCS* uses a dynamic search control strategy to perform an automatic model selection. *MCS* builds trees, which could apply a different model in different regions of the instance space.

Chan and Stolfo [7] presents two schemes for classifier combination: *arbiter* and *combiner*. Both schemes are based on meta learning, where a meta-classifier is generated from a meta data, built based on the predictions of the base classifiers. An arbiter is also a classifier and is used to arbitrate among predictions generated by different base classifiers. The training set for the arbiter is selected from all the available data, using a selection rule. An example of a selection rule is "*Select the examples whose classification the base classifiers cannot predict consistently*". This arbiter, together with an arbitration rule, decides a final classification based on the base predictions. An example of an arbitration rule is "*Use the prediction of the arbiter when the base classifiers cannot obtain a majority*". Later [8], they have extended this framework using *arbiters/combiners* in an hierarchical fashion generating *arbiter/combiner* binary trees.

4.3 Discussion

Earlier results of *boosting* or *bagging* are quite impressive. Using 10 iterations (i.e. generating 10 classifiers) Quinlan [22] reports reductions of the error rate between 10% and 19%. Quinlan argues that these techniques are mainly applicable for unstable classifiers. Both techniques require that the learning system is not stable, to obtain different classifiers when there are small changes in the training set. Under an analysis of bias-variance decomposition of the error [16], the reduction of the error observed with *boosting* or *bagging* is mainly due to the reduction in the variance. As mentioned in Ali et al. [1] "*the number of training examples needed by Boosting increases as a function of the accuracy of the learned model. Boosting could not be used to learn many models on the modest training set sizes used in this paper.*".

Wolpert [25] says that successful implementations of Stacked Generalization is a "*black art*", for classification tasks and the conditions under which stacking works are still unknown. Recently, Ting and Witten [23] have shown that successful stacked generalization requires the use of output class distributions rather than class predictions. In their experiments, only the MLR algorithm (a linear discriminant) was suitable for level-1 generalizer. Cascade Generalization belongs to the family of stacking algorithms. In the experiments described in [14] we have used the *Bias Variance* analysis as a criterion to select algorithms. The experiments suggest that at the top level an algorithm with low *bias*, like a decision tree, should be used.

The main achievement of our proposed method is its ability to merge different models. As such, we get a single model whose components are terms of the base model language. The bias restriction imposed by using single model is relaxed. Cascade gives a single and structured model for the data, and this is a strong advantage over the methods that combine classifiers by voting. Another advantage of Cascade Generalization is related to the use of probability class distributions. Usual learning algorithms produced by the Machine Learning community use categories when classifying examples. Combining classifiers by means of categorical classes looses the strength of the classifier in its prediction. The use of probability class distributions allows us to explore that information.

5 Empirical Evaluation

5.1 The Algorithms

Ali and Pazzani [1] and Tumer and Gosh [24] present empirical and analytical results that show that "*the combined error rate depends on the error rate of individual classifiers and the correlation among them*". They suggest the use of "*radically different types of classifiers*" to reduce the correlation errors. This was our criterion when selecting the algorithms for the experimental work. We use three classifiers that have different behaviors under a *bias-variance* analysis: a naive Bayes, a Linear Discriminant, and a Decision Tree.

5.1.1 Naive Bayes

Bayes theorem allows to optimally predict the class of an unseen example, given a training set. The chosen class is the one that maximizes: $p(C_i|E) = p(C_i)p(E|C_i)/p(E)$. If the attributes are indepen-

dent, $p(E|C_i)$ can be decomposed into the product $p(v_1|C_i) * \dots * p(v_k|C_i)$. Domingos and Pazzani [9] show that this procedure has a surprisingly good performance in a wide variety of domains, including many where there are clear dependencies between attributes. In our reimplementation of this algorithm, the required probabilities are estimated from the training set. In the case of nominal attributes we use counts. Continuous attributes were discretized. This has been found to produce better results than assuming a Gaussian distribution [10, 9]. The number of bins used is a function of the number of different values observed on the training set: $k = \max(1; 2 * \log(nr. \text{different values}))$. This heuristic was used in [10] and elsewhere with good overall results. Missing values were treated as another possible value for the attribute. In order to classify a query point, a *naive Bayes* uses all of the available attributes. Langley [17] refers that *naive Bayes* relies on an important assumption that the variability of the dataset can be summarized by a single probabilistic description, and that these are sufficient to distinguish between classes. From an analysis of *Bias-Variance*, this implies that *naive Bayes* uses a reduced set of models to fit to the data. The result is low variance, but if the data cannot be adequately represented by the set of models, we obtain large bias.

5.1.2 Linear Discriminant

A linear discriminant function is a linear composition of the attributes where the sum of squared differences between class means is maximal relative to the internal class variance. It is assumed that the attribute vectors for the examples of class C_i are independent and follow a certain probability distribution with probability density function f_i . A new point with attribute vector \vec{x} is then assigned to that class for which the probability density function $f_i(\vec{x})$ is maximal. This means that the points for each class are distributed in a cluster centered at μ_i . The boundary separating two classes is a hyper-plane and it passes through the midpoint of the two centers. If there are only two classes, a unique hyper-plane is needed to separate the classes. In the general case of q classes, $q - 1$ hyper-planes are needed to separate them. By applying the linear discriminant procedure described below, we get $q_{node} - 1$ hyper-planes. The equation of each hyper-plane is given by:

$$H_i = \alpha_i + \sum_j \beta_{ij} * x_j \text{ where} \\ \alpha_i = -\frac{1}{2} \mu_i^T S^{-1} \mu_i \text{ and } \beta_i = S^{-1} \mu_i$$

We use a Singular Value Decomposition (SVD) to compute S^{-1} . SVD is numerically stable and is a tool for

detecting sources of collinearity. This last aspect is used as a method for reducing the features of each linear combination. A linear discriminant uses all, or almost all, of the available attributes when classifying a query point. Breiman[2] refers that from an analysis of Bias-Variance, Linear Discriminant is a stable classifier although it can fit a small number of models. It achieves stability by having a limited set of models to fit the data. The result is low variance, but if the data cannot be adequately represented by the set of models, then we obtain large bias.

5.1.3 Decision Tree

Dtree is our version of a decision tree. It uses the standard algorithm to build a decision tree. The splitting criterion is the gain ratio. The stopping criterion is similar to C4.5. The pruning mechanism is similar to the *pessimistic error* of C4.5. *Dtree* uses a kind of smoothing process that usually improves the performance of tree based classifiers. When classifying a new example, the example traverses the tree from the root to a leaf. In *Dtree*, the example is classified taking into account not only the class distribution at the leaf, but also all class distributions of the nodes in the path. That is, all nodes in the path contribute to the final classification. Instead of computing class distribution for all paths in the tree at classification time, as it is done, for instance, in Buntine [6], *Dtree* computes a class distribution for all nodes when growing the tree. This is done recursively, taking into account class distributions at the current node and at the predecessor of the current node, using the formula:

$$P(C_i|e_n, e) = P(C_i|e_n) \frac{P(e|e_n, C_i)}{P(e|e_n)}$$

where $P(e|e_n)$ is the probability that one example that falls at *Node_n* goes to *Node_{n+1}*, and $P(e|e_n, C_i)$ is the probability that one example from class C_i goes from *Node_n* to *Node_{n+1}* [21]. This recursive formulation, allows *Dtree* to compute efficiently the required class distributions on the fly. The smoothed class distributions have influence on the pruning mechanism and on the treatment of missing values. It is the most relevant difference from C4.5.

A decision tree uses a subset of the available attributes to classify a query point. Kohavi and Wolpert [16], Breiman [2, 3] among other researchers, note that decision trees are unstable classifiers. Small variations on the training set can cause large changes in the resulting predictors. They have high variance but they can fit any kind of data: the bias of a decision tree is low.

5.1.4 Local Cascade Generalization Algorithms

All the implemented *Local Cascade Generalization algorithms* are based on *Dtree*. That is they use exactly the same splitting criteria, stopping criteria, pruning mechanism, etc. Moreover they share many minor heuristics that individually are too small to mention, but collectively can make difference.

At each decision node, *CGLtree* applies the *Linear discriminant* describe above, while *CGBtree* applies the *naive Bayes* algorithm. *CGBLtree* applies the *Linear discriminant* to the *ordered* attributes and the *naive Bayes* to the *categorical* attributes. In order to prevent *overfitting* the construction of new attributes is constrained to a depth of 5. In addition, the level of pruning is greater than the level of pruning in *Dtree*.

5.2 The Datasets

We have chosen 17 data sets from the UCI repository. All of them were previously used in other comparative studies. Evaluation was done using a 10 fold *stratified Cross Validation* (CV). Datasets were permuted once before the CV procedure. All algorithms were used with the default settings. At each iteration of CV, all algorithms were trained on the same training partition of the data. Classifiers were also evaluated on the same test partition of the data. Comparisons between algorithms were performed using *t-paired* tests with significance level set at 95%.

Table 1 presents the data sets characteristics, the error rate, and standard deviation of each base classifier. Relative to each algorithm, a $+(-)$ sign on the first column means that the error rate of this algorithm, is significantly better (worse) than *Dtree*. The error rate of *C5.0* is presented for reference. These results provide an evidence, once more, that no single algorithm is better overall.

5.3 Local Cascade Generalization

Table 2a presents the results of local Cascade Generalization. Each column corresponds to a Cascade Generalization algorithm. Each algorithm is compared against its components using *t-paired* tests. For example, *CGLtree* is compared against *Dtree* and *Discrim*. A $+(-)$ sign means that the error rate of the composite model is, with statistical significance, higher (lower) than the respective component model. The trend on these results shows a clear improvement over the base classifiers. We never observe degradation

on the error rate of a composite model in relation to all the components. In some cases there is a significant increase of performance comparing to all the components. For example *CGBLtree* improves in 2 datasets over the 3 components, and in 5 datasets over 2 components.

Table 2b presents the results of *C5.0 boosting* with the default parameter of 10, that is aggregating over 10 trees, and *Stacked Generalization* as it is defined in [23]. That is, the *level₀* classifiers are *C4.5* and *Bayes*, and the *level₁* classifier is *Discrim*. The attributes for the *level₁* data are the probability class distributions, obtained from the *level₀* classifiers using a 5 stratified cross validation. Both *Boosting* and *Stacked* are compared against *CGBLtree*, using *t-paired* tests with the significance level set to 95%. A $+(-)$ sign means that *Boosting* or *Stacked* performs significantly better (worse) than *CGBLtree*. In this study, *CGBLtree* performs significantly better than *Stacked*, in 5 datasets and never performs worse. Comparing with *C5.0 Boosting*, *CGBLtree* significantly improves in 4 datasets and loses in 3 datasets. The improvement observed with *Boosting* is mainly due to the reduction of the *variance* component of the error rate while, in *Cascade* algorithms, the improvement is mainly due to the reduction on the *bias*. We intend, in a near future, to *boost CGBLtree*.

Another dimension for comparisons involves measuring the number of leaves. This corresponds to the number of different regions into which the instance space is partitioned by the algorithm. In almost all datasets³, any Cascade tree splits the instance space into half of the regions needed by *Dtree* or *C5.0*. This is a clear indication that Cascade models capture better the underlying structure of the data.

6 Conclusions

This paper presents a new methodology for classifier combination. The basic idea of Cascade Generalization consists of a reformulation of the input space by means of insertion of new attributes. A base classifier computes the new attributes. Each new attribute is the instantiation of $P(C_i|\vec{x})$ given by the predictor function generated by the base classifier on this example. In this sense, the new attributes are terms, or functions, in the representational language of the base classifier. This constructive step acts as a way of extending the description language of the high level

³Except on Monks-2 dataset, where both *Dtree* and *C5.0* produce a tree with only one leaf.

Dataset	Class	Nr.Ex.	Types	Dtree	C5.0	Bayes	Discrim
Australian	2	690	8 Ord,6 Cont	14.37 \pm 6.18	13.63 \pm 4.36	15.07 \pm 3.76	14.05 \pm 5.23
Balance	3	625	4 Cont	21.91 \pm 4.63	21.92 \pm 4.93	30.08 \pm 7.01	13.14 \pm 2.46
Breast(W)	2	699	9 Ord	5.84 \pm 4.64	5.42 \pm 4.08	2.43 \pm 2.52	4.27 \pm 4.58
Diabetes	2	768	8 Cont	25.14 \pm 5.78	23.69 \pm 6.48	24.62 \pm 4.58	22.92 \pm 4.97
German	2	1000	17 Ord,7 Cont	28.70 \pm 4.30	29.10 \pm 2.81	27.60 \pm 5.15	23.50 \pm 5.54
Glass	6	213	9 Cont	31.85 \pm 7.61	32.30 \pm 10.19	46.35 \pm 10.93	36.78 \pm 8.07
Heart	2	270	6 Ord,7 Cont	25.16 \pm 9.84	22.96 \pm 8.69	15.93 \pm 8.56	15.93 \pm 4.29
Ionosphere	2	351	33 Cont	8.54 \pm 5.80	9.66 \pm 3.47	11.07 \pm 7.76	14.26 \pm 4.68
Iris	3	150	4 Cont	4.67 \pm 5.48	4.67 \pm 4.50	4.00 \pm 4.66	2.00 \pm 3.22
Monks-1	2	432	6 Ord	6.33 \pm 7.45	0.00 \pm 0.00	25.07 \pm 5.96	33.39 \pm 10.06
Monks-2	2	432	6 Ord	32.90 \pm 0.63	32.86 \pm 0.65	49.32 \pm 8.50	33.32 \pm 1.60
Monks-3	2	432	6 Ord	0.00 \pm 0.00	0.00 \pm 0.00	2.79 \pm 2.42	22.89 \pm 8.96
Satimage	6	6435	36 Cont	13.35 \pm 1.51	13.53 \pm 1.57	19.55 \pm 1.48	15.91 \pm 1.49
Segment	7	2310	18 Cont	3.64 \pm 1.13	3.38 \pm 1.34	10.22 \pm 0.74	8.18 \pm 0.83
Vehicle	4	846	18 Cont	28.11 \pm 4.87	27.27 \pm 5.48	37.70 \pm 2.18	22.34 \pm 2.87
Waveform	3	2581	21 Cont	23.38 \pm 3.40	24.88 \pm 2.94	18.52 \pm 2.24	15.15 \pm 1.86
Wine	3	178	13 Cont	6.66 \pm 6.32	7.19 \pm 7.44	2.22 \pm 3.88	0.56 \pm 1.76
Mean of error rate				16.50	16.02	20.15	17.56
Mean nr. Leaves				45.6	51.3		

Table 1: Data Characteristics and Results of Base Classifiers

Dataset	CGLtree	CGBtree	CGBLtree	C5Boost	Stacked
Australian	14.354 \pm 4.77	14.499 \pm 3.76	14.058 \pm 4.80	13.337 \pm 3.33	13.766 \pm 4.47
Balance	+ + 7.016 \pm 2.68	+ + 6.704 \pm 3.64	+ + + 7.016 \pm 2.68	- 20.184 \pm 4.17	- 12.309 \pm 3.63
Breast (W)	+ 3.280 \pm 2.59	+ 2.712 \pm 2.27	+ 3.280 \pm 2.68	- 3.135 \pm 3.20	- 2.427 \pm 2.52
Diabetes	23.565 \pm 3.12	26.693 \pm 5.87	23.565 \pm 3.12	24.728 \pm 5.46	22.657 \pm 5.42
German	24.700 \pm 4.19	27.100 \pm 5.48	- 25.300 \pm 5.25	23.200 \pm 2.35	24.800 \pm 4.24
Glass	33.866 \pm 9.26	+ 27.004 \pm 7.51	+ 33.866 \pm 9.26	25.020 \pm 10.09	35.753 \pm 6.20
Heart	+ 17.037 \pm 5.58	+ 16.667 \pm 6.11	+ 17.037 \pm 5.58	- 19.630 \pm 9.25	- 16.667 \pm 8.24
Ionosphere	11.363 \pm 4.32	9.369 \pm 5.12	11.363 \pm 4.32	+ 5.947 \pm 3.06	10.758 \pm 7.33
Iris	2.667 \pm 3.44	4.000 \pm 4.66	2.667 \pm 3.44	- 5.333 \pm 4.22	- 4.667 \pm 3.22
Monks-1	+ 2.976 \pm 4.43	- + 14.372 \pm 8.69	+ + 2.565 \pm 3.77	0.000 \pm 0.00	0.682 \pm 2.16
Monks-2	33.335 \pm 5.81	+ + 13.874 \pm 6.97	+ + + 11.120 \pm 5.36	- 36.353 \pm 5.87	- 32.865 \pm 0.65
Monks-3	+ 0.698 \pm 1.12	+ 0.465 \pm 0.47	+ + 0.465 \pm 1.47	0.000 \pm 0.00	- 2.072 \pm 2.01
Satimage	+ 12.385 \pm 1.44	+ + 11.673 \pm 1.25	+ + 12.385 \pm 1.44	+ 9.062 \pm 1.07	- 13.303 \pm 1.63
Segment	+ 3.853 \pm 1.22	+ 4.416 \pm 1.47	+ + 3.853 \pm 1.21	+ 1.905 \pm 1.05	3.420 \pm 1.35
Vehicle	+ 21.025 \pm 3.08	+ 28.844 \pm 3.88	+ + 21.025 \pm 3.08	- 24.922 \pm 3.71	- 27.731 \pm 5.06
Waveform	+ 16.351 \pm 1.68	+ 16.004 \pm 2.78	+ 16.351 \pm 1.68	17.980 \pm 1.86	16.429 \pm 1.50
Wine	+ 0.556 \pm 1.76	3.403 \pm 3.94	+ 0.556 \pm 1.76	2.222 \pm 2.87	2.778 \pm 3.93
Mean error rate	13.47	13.40	12.15	13.70	14.30
Mean nr.leaves	23.9	23.7	22.9		

Table 2: Results of (a)Local Cascade Generalization (b)Boosting and Stacked

classifiers. The number of new attributes is equal to the number of classes, and for each example, they are computed as the conditional probability of the example belonging to $class_i$ given by the base classifier.

Cascade Generalization can be applied *locally* by any learning algorithm that uses a divide-conquer strategy. As pointed by several researchers, successful combination of classifiers requires different syntactic models. We have chosen, for the implementation of *Local Cascade Generalization* algorithms, three algorithms that have very different behavior from a *bias-variance* analysis: as high level classifier we use a decision tree and as low level classifier we use a naive Bayes, giving *CG-Btree* and a Linear Discriminant, giving *CGLtree*. At each decision node a constructive step is performed by applying the base classifier. The new axis incorporates new knowledge provided by the base classifiers. The

bias restriction imposed by using single model classes is relaxed in the directions given by the base classifiers. It is this kind of synergy among classifiers that Cascade explores.

There are two main issues that differentiate Cascade from other previous methods on multiple models. The first one is related to its ability to be applied *locally* merging different models. We get a single model whose components are terms of the base model language, extending the high level model language. Cascade gives a single structured model for the data, and in this way is more adapted to capture insights about problem structure. The second point is related to the use of probability class distributions. Using these probabilities allows the system to use information about the strength of the classifier. This is very useful information, particularly when combining predictions of

classifiers. We have shown that this methodology can improve the accuracy of the base classifiers, competing well with other methods for combining classifiers, preserving the ability to provide a single albeit structured model for the data.

Acknowledgements

Gratitude is expressed to the support given by the FEDER and PRAXIS XXI projects and the Plurianual support attributed to LIACC. Thanks to P. Brazdil, colleagues from LIACC, and the anonymous reviewers for the valuable comments.

References

- [1] K. Ali and M. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, Vol. 24, No. 1, 1996.
- [2] L. Breiman. Bias, variance, and arcing classifiers. Technical report 460, Statistics Department, University of California, 1996.
- [3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group., 1984.
- [4] C. Brodley. Recursive automatic bias selection for classifier construction. *Machine Learning*, 20, 1995.
- [5] C. Brodley and P. Utgoff. Multivariate trees. *Machine Learning*, 19, 1995.
- [6] Wray Buntine. *A theory of Learning Classification Rules*. PhD thesis, University of Sydney, 1990.
- [7] P. Chan and S. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In A. Prieditis and S. Russel, editors, *Machine Learning Proc of 12th International Conference*. Morgan Kaufmann, 1995.
- [8] P. Chan and S. Stolfo. Learning arbiter and combiner trees from partitioned data for scaling machine learning. In *KDD 95*, 1995.
- [9] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In L. Saitta, editor, *Machine Learning Proc. of 13th International Conference*. Morgan Kaufmann, 1996.
- [10] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russel, editors, *Machine Learning Proc. of 12th International Conference*. Morgan Kaufmann, 1995.
- [11] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1990.
- [12] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In L. Saitta, editor, *Machine Learning Proc of 13th International Conference*. Morgan Kaufmann, 1996.
- [13] J. Gama. Probabilistic linear tree. In D. Fisher, editor, *Machine Learning Proc. of the 14th International Conference*. Morgan Kaufmann, 1997.
- [14] J. Gama. Combining classifiers by constructive induction. In C. Nedellec and C. Rouveirol, editors, *Machine Learning ECML-98*. Springer Verlag, 1998.
- [15] G. John. Robust linear discriminant trees. In D. Fisher, editor, *Learning from Data: Artificial Intelligence and Statistics V*. Springer Verlag, 1996.
- [16] R. Kohavi and D. Wolpert. Bias plus variance decomposition for zero-one loss function. In L. Saitta, editor, *Machine Learning Proc. of 13th International Conference*. Morgan Kaufmann, 1996.
- [17] P. Langley. Induction of recursive bayesian classifiers. In P. Brazdil, editor, *Machine Learning: ECML-93*. LNAI 667, Springer Verlag, 1993.
- [18] Pat Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.
- [19] S. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 1994.
- [20] J. Ortega. Exploiting multiple existing models and learning algorithms. In *AAAI 96 - Workshop in Induction of Multiple Learning Models*, 1995.
- [21] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.
- [22] R. Quinlan. Bagging, boosting and c4.5. In *Procs. 13th American Association for Artificial Intelligence*. AAAI Press, 1996.
- [23] K.M. Ting and I.H. Witten. Stacked generalization: when does it work ? In *Procs. International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1997.
- [24] K. Tumer and J. Ghosh. Classifier combining: analytical results and implications. In *AAAI 96 - Workshop in Induction of Multiple Learning Models*, 1995.
- [25] D. Wolpert. Stacked generalization. In Pergamon Press, editor, *Neural Networks Vol.5*, 1992.

A Learning Rate Analysis of Reinforcement Learning Algorithms in Finite-Horizon

Frédéric Garcia
 INRA/BIA, Auzeville BP 27
 31326 Castanet Tolosan cedex
 France
 fgarcia@toulouse.inra.fr

Seydina M. Ndiaye
 INRA/BIA, Auzeville BP 27
 31326 Castanet Tolosan cedex
 France
 ndiaye@toulouse.inra.fr

Abstract

Many reinforcement learning algorithms, like Q-Learning or R-Learning, correspond to adaptative methods for solving Markovian decision problems in infinite-horizon when no model is available. In this article we consider the particular framework of non-stationary finite-horizon Markov Decision Processes. After establishing a relationship between the finite-horizon total reward criterion and the average-reward criterion in finite-horizon, we define Q_H -Learning and R_H -Learning for finite-horizon MDPs. Then we introduce the Ordinary Differential Equation (ODE) method to conduct a learning rate analysis of Q_H -Learning and R_H -Learning. R_H -Learning appears to be a version of Q_H -Learning with matrix-valued stepsizes, the corresponding gain matrix being very close to the optimal matrix which results from the ODE analysis. Experimental results confirm that performance hierarchy.

1 Introduction

The search for optimal policies in Markov Decision Processes has been deeply studied according to different optimality criteria and has led to the definition of the well known Bellman optimality equations, and dynamic programming algorithms [Puterman, 1994]. Most Reinforcement Learning (RL) algorithms that have been recently developed [Kaelbling et al., 1996, Bertsekas and Tsitsiklis, 1996] take a stochastic optimization approach to solve these optimality equations, by directly learning the optimal policies from iterated observations of rewards and state transitions, without

a priori knowledge about the system.

In this paper we consider the case of non stationary Markov decision problems in a finite horizon. Despite being an accurate modelling of many applications concerning the management of industrial production systems, finite-horizon MDPs have not been yet specifically considered in reinforcement learning. This article is a first attempt to fill this gap.

Our work relies on two parts. First we propose a reformulation of the two main classical optimality criteria, expected total reward criterion and average expected reward criterion, given the finite-horizon assumption. After establishing an equivalence between them, we conclude that it is possible to use the two adapted reinforcement learning algorithms, Q_H -Learning and R_H -Learning, to learn optimal policies for non stationary finite-horizon MDPs.

Secondly, we conduct an analysis of the respective rates of convergence of Q_H -Learning and R_H -Learning. Surprisingly, R_H -Learning appears to be a version of Q_H -Learning with matrix-valued stepsizes. Furthermore, the ordinary differential equation (ODE) method enables to determine a theoretical optimal matrix-valued gain, and it appears that the gain corresponding to R_H -Learning is numerically and structurally very close to that optimal gain. The experimental study we conducted confirms these results: in most situations we tested, R_H -Learning performs better than Q_H -Learning, and the implementation of the optimal matrix-valued gain defines a reinforcement algorithm that surpasses R_H -Learning.

2 Reinforcement Learning in Finite-Horizon

2.1 Non-Stationary MDP in Finite-Horizon

The majority of reinforcement learning algorithms solve stationary infinite-horizon Markov Decision Problems. Given a state-space S and an action-space A , the dynamic of a Markov decision process is characterized as follows: at each time step $t \in T$, the execution of action $a_t \in A$ in state $x_t \in S$ leads to the new state $x_{t+1} \in S$ with a probability $p(x_{t+1} | x_t, a_t)$, and to the instantaneous reward $r(x_t, a_t)$.

A Markov Decision Problem is defined by adding to that process a performance criterion to maximize over a set of decisional policies. This criterion is a measure of the expected sum of the rewards along a trajectory, and policies are functions that indicate the action a_t to execute given informations about the past trajectory at time t . For stationary infinite-horizon Markov decision problems, most of the performance criteria lead to the existence of stationary optimal policies, i.e. functions π that map states in S to actions in A .

In finite-horizon problems, trajectories are sequences of exactly N transitions, with $T = \{1, \dots, N\}$. The performance criterion considered in that case is the *finite total expected reward* criterion $V^\pi(x) = E_\pi[r(x_1, a_1) + r(x_2, a_2) + \dots + r(x_N, a_N) | x_1 = x]$ where $x \in S$ and E_π is the expected value given the policy π .

When dealing with finite-horizon MDPs, the stationary assumption cannot be considered anymore. A first reason is that even for stationary finite-horizon MDP models (time-independent spaces S and A , transition probabilities $p()$ and rewards $r()$), optimal policies are no longer stationary, and are functions of $T \times S$ into A : $t, x \xrightarrow{\pi} \pi(t, x)$ [Puterman, 1994].

More practically, in most of the problems of industrial production process control that lead to finite-horizon MDPs, the main cause of non-stationarity of optimal policies is the non-stationarity of the MDP model itself: it is very common to have different state-spaces, decision-spaces, transition probabilities and reward values at each decision step. In order to take into account this characteristic, we consider the following formal model of finite-horizon MDP: 1) to each time step $i \in T = \{1, 2, \dots, N\}$ is associated a finite state space S_i and a finite decision space A_i ; 2) for each step $i \in \{1, 2, \dots, N-1\}$, the execution of action

$a_i \in A_i$ from the state $x_i \in S_i$ leads to the new state $x_{i+1} \in S_{i+1}$ with a probability $p_i(x_{i+1} | x_i, a_i)$, and with the instantaneous reward $r_i(x_i, a_i)$; 3) at the last decision step, the system receives a reward $r_N(x_N, a_N)$ after the execution of a_N in x_N , and stops.

For this particular kind of MDP a policy π can be decomposed into a set $\{\pi_1, \pi_2, \dots, \pi_N\}$ of policies $\pi_i : S_i \rightarrow A_i$. For each decision step, a value function associated to π is defined as $V_i^\pi(x) = E_\pi[\sum_{t=i}^N r_t(x_t, \pi_t(x_t)) | x_i = x]$.

We say a policy π is optimal if it maximizes the value function V_1^π on S_1 . For this criterion, the classical Bellman optimality equations that characterize optimal policies are

$$V_i^*(x) = \max_a \left\{ r_i(x, a) + \sum_{y \in S_{i+1}} p_i(y | x, a) V_{i+1}^*(y) \right\} \quad (1)$$

for all $x \in S_i$, $i \in \{1, \dots, N\}$ and $V_{N+1}^* = 0$ [Puterman, 1994]. Then $\pi_i^*(x) = \operatorname{argmax}_a \{r_i(x, a) + \sum_{y \in S_{i+1}} p_i(y | x, a) V_{i+1}^*(y)\}$. This optimality equation has a single solution $V^* = \{V_1^*, \dots, V_N^*\}$, that can be easily obtained by a dynamic programming algorithm in $O(N n_A n_S^2)$ complexity (for state spaces S_i and decision spaces A_i of constant size n_S and n_A) when transition probabilities and reward function are known [Puterman, 1994]. The associated learning problem is to adaptatively estimate the optimal value functions V_i^* and the corresponding policies π_i^* , from observed transitions and rewards when the Markov decision process is not known.

2.2 Q_H -Learning in Finite-Horizon

Q-Learning [Watkins, 1989] is based on the rewriting of the Bellman optimality equation, replacing the $V^\pi(x)$ value function of a policy by a new function $Q^\pi(x, a)$: for all $x \in S_i$, $a \in A_i$ $Q_i^\pi(x, a) = r_i(x, a) + \sum_{y \in S_{i+1}} p_i(y | x, a) V_{i+1}^\pi(y)$.

We have $V^\pi(x) = Q^\pi(x, \pi(x))$, and the optimality equation becomes

$$Q_i^*(x, a) = r_i(x, a) + \sum_{y \in S_{i+1}} p_i(y | x, a) \max_b Q_{i+1}^*(y, b),$$

for all $x \in S_i$ and $a \in A_i$. Then $V^*(x) = \max_a Q^*(x, a)$ and $\pi^*(x) = \operatorname{argmax}_a Q^*(x, a)$.

Q-Learning is a reinforcement learning algorithm allowing the iterative generation of the solution Q^* and

the optimal policies π^* . The algorithm consists in updating at each iteration n the estimation Q_n of the value function Q^* , from the current observed transition and reward $\langle x_n, a_n, y_n, r_n \rangle$. Q-Learning is a natural candidate for solving finite-horizon MDPs. It is indeed easy to transform an N-step non-stationary MDP into an infinite-horizon process, by adding an artificial final absorbing state x_{abs} , that is reward-free and such that all actions a_N in A_N lead with probability 1 to x_{abs} (figure 1).

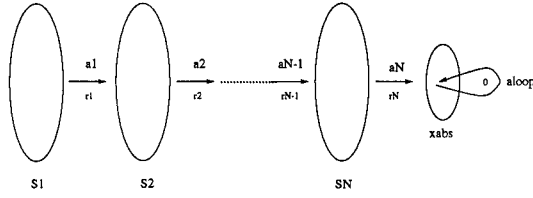


Figure 1: infinite process with absorbing state

Hence the first reinforcement learning algorithm we propose for finite horizon MDPs is:

Finite-Horizon Q_H -Learning

Observe $\langle x_n, a_n, y_n, r_n \rangle$

Update

$$Q_{n+1}(x, a) = Q_n(x, a) + \alpha_n(x, a) \cdot e_n \quad (2)$$

with $e_n =$

$$\begin{cases} r_n + \max_b Q_n(y_n, b) - Q_n(x, a) & \text{if } (x, a) = (x_n, a_n), x_n \in S_i, i < N \\ r_n - Q_n(x, a) & \text{if } (x, a) = (x_n, a_n), x_n \in S_N \\ 0 & \text{otherwise} \end{cases}$$

If $x_n \notin S_N$ set $x_{n+1} = y_n$;

otherwise choose randomly x_{n+1} in S_1 .

If $x_{n+1} \in S_j$ select a_{n+1} in A_j

In this algorithm $Q_0(x, a) = 0$ and $\alpha_n(x, a)$ are small learning rates decaying over time. The state exploration is classically determined by the dynamic of the process (that is, $x_{n+1} = y_n$), until the last decision step is reached and we restart a new trajectory by choosing randomly a new initial state in S_1 . The specific learning rule for S_N is equivalent to directly setting $V_{N+1}^*(x_{abs}) = Q_{N+1}^*(x_{abs}, a_{loop}) = 0$. The action selection is as usual based on an exploration function.

Let us assume that each pair (x, a) in $S_i \times A_i$ is visited an infinite number of times, and that $\sum_n \alpha_n(x, a) = \infty$ and $\sum_n \alpha_n^2(x, a) < \infty$. The convergence of Q-Learning [Watkins and Dayan, 1992,

Jaakkola et al., 1994, Tsitsiklis, 1994] in case of no-discounting ($\gamma = 1$) and with the presence of reward-free absorbing states proves that this finite-horizon Q_H -Learning algorithm will converge in probability 1 towards the optimal value function: $\forall x \in S_i, a \in A_i, \lim_{n \rightarrow \infty} Q_n(x, a) = Q_i^*(x, a)$ a.s. with $V_i^*(x) = \max_{a \in A_i} Q_i^*(x, a)$.

2.3 R_H -Learning and the average-reward criterion

The average reward criterion was introduced in Reinforcement Learning by Schwartz through the R-Learning algorithm [Schwartz, 1993]. It has been studied since then by many researchers [Singh, 1994, Ok and Tadepalli, 1996, Mahadevan, 1996b]. The goal is to search for *gain-optimal* policies that maximize the expected payoff per step, which is a very natural measure of optimal acting:

$$\rho^\pi(x) = \lim_{n \rightarrow \infty} E_\pi \left[\frac{1}{n} \sum_{t=1}^n r_t \mid x_1 = x \right].$$

For the particular case of *unichain* MDPs (that is, for all policy π , the Markov chain $\{x_n\}_n$ contains a single recurrent class of states, and a possibly empty set of transient states), the average reward associated to each policy is independent of the state: $\rho^\pi(x) = \rho^\pi(y) = \rho^\pi$. For simplicity reasons, most of the results concerning average reward criterion in Reinforcement Learning have been established with this unichain assumption [Mahadevan, 1996b].

A more selective optimality criterion can be defined. It is based on a new value function U^π of a policy π , called *bias value* [Puterman, 1994]. For all state $x \in S$ we have

$$U^\pi(x) = \lim_{n \rightarrow \infty} E_\pi \left[\sum_{t=1}^n (r_t - \rho^\pi) \mid x_1 = x \right].$$

A policy π^* is said to be *bias-optimal* (or *T-optimal* in [Schwartz, 1993]) if it is gain-optimal, and if $U^{\pi^*}(x) \geq U^\pi(x)$ for all x and all policy π .

The existence of optimal stationary policies for gain and bias optimality has been shown [Puterman, 1994]. For all unichain MDPs, there exists a pair (U^*, ρ^*) solution of the Bellman equation for the average criterion:

$$U^*(x) + \rho^* = \max_a \left(r(x, a) + \sum_y p(y \mid x, a) U^*(y) \right), \quad (3)$$

for all $x \in S$, such that the average reward of the policy π^* that maximizes the right-hand side of (3) is the optimal average reward ρ^* . Furthermore, if (U^*, ρ^*) is also a solution of (3), then $\rho^* = \rho^{*'}$. The solutions U^* of (3) are not unique, since for each solution (U^*, ρ^*) , the pair $(U^* + k, \rho^*)$ is also a solution (one can show that this is a complete characterization of the set of solutions for unichain MDP models [Puterman, 1994]).

That last remark shows that (3) is not sufficient to produce bias optimal policies. Another optimality equation, based on a third notion of value function called *bias offset*, is generally required [Puterman, 1994, Mahadevan, 1996a].

In order to adapt the average-reward criterion to finite-horizon MDPs, we first transform the initial process $S_1 \rightarrow S_N$ into a new infinite process $S_1 \rightarrow S_N \cup S_1$. The natural solution we propose is to close artificially the loop between S_N and S_1 by adding a uniform transition: $\forall x \in S_N, \forall a \in A_N, \forall y \in S_1, p_N(y | x, a) = \frac{1}{n_{S_1}}$ (figure 2).

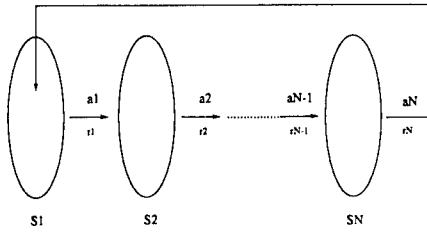


Figure 2: infinite process with looping on S_1

For the new MDP $S_1 \rightarrow S_N \cup S_1$ we proved the following proposition [Garcia and Ndiaye, 1998]:

Proposition 1 For the cycling process $S_1 \rightarrow S_N \cup S_1$, for all policy π ,

$$\forall x \in S_1 \quad \rho^\pi(x) = \frac{1}{N} \frac{1}{n_{S_1}} \sum_{x_1 \in S_1} V_1^\pi(x_1) = \rho^\pi$$

$$\forall x \in S_i, i=1, \dots, N \quad U^\pi(x) = V_i^\pi(x) - (N - i + 1)\rho^\pi, \\ \sum_{x_1 \in S_1} U^\pi(x_1) = 0.$$

The first aspect of this result, the state independence of ρ^π , is not surprising since the looping $S_N \cup S_1$ transforms the original MDP into a *unichain* process. More interesting are the next equalities; the

second one establishes an equivalence between the bias-value function, the average reward and the value function in finite-horizon, and the last one completely determines this bias-value function. From that properties we proved the following theorem [Garcia and Ndiaye, 1998]:

Theorem 1 If (U^*, ρ^*) is a solution of average-reward Bellman equation (3) for $S_1 \rightarrow S_N \cup S_1$ with the constraint $\sum_{x_1 \in S_1} U^*(x_1) = 0$, and if π^* is an associated gain-optimal policy, then the value functions $V_i^*(x) = U_i^*(x) + (N - i + 1)\rho^*$ are solutions of finite-horizon Bellman equation (1), and π^* is a policy that maximizes $V_i^\pi(x)$ for $x \in S_i, i = 1, \dots, N$.

That result shows that there is an equivalence between the finite-horizon and average-reward criteria, and a solution of (3) necessary leads to a solution of (1). The following corollary characterizes more deeply this equivalence [Garcia and Ndiaye, 1998].

Corollary 1 If (U^*, ρ^*) is solution of (3) for $S_1 \rightarrow S_N \cup S_1$ with $\sum_{x_1 \in S_1} U^*(x_1) = 0$, then (U^*, ρ^*) also defines a bias-optimal solution.

From these results, it appears natural to use R-Learning for solving finite-horizon MDPs. The second reinforcement learning algorithm we propose, called $R_{\mathcal{H}}$ -Learning, is an adaptation of R-Learning with an update rule for states in S_N that directly integrates the final condition $R_N^*(x, a) = r_N(x, a) - \rho^\pi$ for $x \in S_N, a \in A_N$:

Finite-Horizon $R_{\mathcal{H}}$ -Learning

Observe $\langle x_n, a_n, y_n, r_n \rangle$

Update

$$R_{n+1}(x, a) = R_n(x, a) + \alpha_n(x, a) \cdot e_n \\ \rho_{n+1} = \rho_n + \beta_n \cdot e'_n \\ e_n = \begin{cases} r_n - \rho_n + \max_b R_n(y_n, b) - R_n(x, a) \\ \text{if } (x, a) = (x_n, a_n) \text{ } x_n \in S_i, i < N \\ r_n - \rho_n - R_n(x, a) \\ \text{if } (x, a) = (x_n, a_n) \text{ } x_n \in S_N \\ 0 \text{ otherwise} \end{cases} \\ e'_n = \begin{cases} r_n - \rho_n + \max_b R_n(y_n, b) - R_n(x_n, a_n) \\ \text{if } x_n \in S_i, i < N \text{ } a_n = \pi_n(x_n) \\ r_n - \rho_n - R_n(x_n, a_n) \\ \text{if } x_n \in S_N \text{ } a_n = \pi_n(x_n) \\ 0 \text{ otherwise} \end{cases}$$

If $x_{n+1} = y_n \in S_j$ select a_{n+1} in A_j

3 A learning rate analysis of $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning

The simulations we conducted from a random finite-MDP generator (see [Garcia and Ndiaye, 1998] and section 4) have shown experimentally that $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning always converge to an optimal policy, and that $R_{\mathcal{H}}$ -Learning is most of the time faster than $Q_{\mathcal{H}}$ -Learning. However, it still does not exist any definitive theoretical results about the convergence of R -Learning-like algorithms, with a mixed iteration on R_n and ρ_n .

The aim of this section is to introduce a comparison of the respective learning rates of convergence of $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning. The analysis we propose below is made possible by an original equivalent transformation of $R_{\mathcal{H}}$ -Learning into a new reinforcement algorithm, the form of which is closer to $Q_{\mathcal{H}}$ -Learning. We first present that transformation.

3.1 An equivalent formulation of $R_{\mathcal{H}}$ -Learning

Just consider the second equation of Proposition 1. It sets a direct relation between the value functions U^π and V^π of a policy π , that can be directly translated in terms of functions R^π and Q^π : $\forall x \in S_i, \forall a \in A_i$ $R_i^\pi(x, a) = Q_i^\pi(x, a) - (N - i + 1)\rho^\pi$. From that observation we propose to transform the iteration on R_n in the $R_{\mathcal{H}}$ -Learning algorithm by an iteration on Q_n . With this aim, we define the new series $\{Q_n\}_n$:

$$\forall x \in S_i, \forall a \in A_i \quad Q_n(x, a) = R_n(x, a) + (N - i + 1)\rho_n \quad (4)$$

with $\{R_n\}_n$ and $\{\rho_n\}_n$ the two series of the $R_{\mathcal{H}}$ -Learning algorithm. That transformation leads to the following equivalent reinforcement algorithm:

Finite-Horizon $R_{\mathcal{H}}$ -Learning - Q formulation —

$$\begin{aligned} Q_{n+1}(x, a) &= Q_n(x, a) + \gamma_n(x, a) \cdot e_n \\ \rho_{n+1} &= \rho_n + \beta_n \cdot e_n \text{ if } a_n = \pi_n(x_n) \\ e_n &= \begin{cases} r_n + \max_b Q_n(x_n, b) - Q_n(x_n, a_n) & \text{if } x_n \in S_i, i < N \\ r_n - Q_n(x_n, a_n) & \text{if } x_n \in S_N \end{cases} \\ \gamma_n(x, a) &= \begin{cases} \alpha_n(x, a) + (N - i + 1)\beta_n & \text{if } (x, a) = (x_n, a_n), x_n \in S_i, a_n = \pi_n(x_n) \\ \alpha_n(x, a) & \text{if } (x, a) = (x_n, a_n), a_n \neq \pi_n(x_n) \\ (N - i + 1)\beta_n & \text{if } (x, a) \neq (x_n, a_n), x \in S_i, a_n = \pi_n(x_n) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (5)$$

As we can see, the two series $\{Q_n\}_n$ and $\{\rho_n\}_n$ are now decoupled. Furthermore, since $\pi_n(x) = \operatorname{argmax}_a R_n(x, a) = \operatorname{argmax}_a (Q_n(x, a) - (N - i + 1)\rho_n) = \operatorname{argmax}_a Q_n(x, a)$, the $\{\rho_n\}_n$ iteration is even not necessary to determine the current policy π_n .

Hence the two algorithms $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning in finite-horizon can be considered as two different updating rules of the same value function Q . More precisely, the main difference between $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning can now be clearly associated to the number of components $Q_n(x, a)$ which are modified at each iteration of the algorithm. In $Q_{\mathcal{H}}$ -Learning, we only update the component $Q_n(x_n, a_n)$. In $R_{\mathcal{H}}$ -Learning, if $(x, a) \neq (x_n, a_n)$, $Q(x, a)$ can still be updated if the action a_n corresponds to a greedy action for the state x_n in the policy π_n .

3.2 Reinforcement Learning and the ODE method

Now that we have seen that $R_{\mathcal{H}}$ -Learning is a parallel version of $Q_{\mathcal{H}}$ -Learning, we intend to compare their respective rates of convergence. The theoretical tool we have chosen is the Ordinary Differential Equation (ODE) method recently introduced in reinforcement learning [Bertsekas and Tsitsiklis, 1996, Kushner and Yin, 1997]. The ODE method results from the combination of dynamical systems and stochastic approximation techniques. The classical theory of stochastic approximation introduced by Robbins and Monro [Robbins and Monro, 1951] concerns the analysis of adaptive stochastic algorithms

$$\theta_{n+1} = \theta_n + \gamma_n H(\theta_n, X_{n+1}) \quad (6)$$

where θ_n is the parameter vector, and X_n the input random vector bringing some information on θ_n at time n . The application of this theory to the domain of Reinforcement Learning has led to general proofs of convergence for Q -Learning or $TD(\lambda)$ [Jaakkola et al., 1994, Tsitsiklis, 1994]. The ODE method was initially proposed by Ljung [Ljung, 1977], and then has been the source of many works, as in [Kushner and Clark, 1978, Benaïm, 1996]. It consists in the introduction of the averaged differential equation $\frac{d\theta}{dt} = \bar{H}(\theta)$ where $\bar{H}(\theta) = \lim_{n \rightarrow \infty} E[H(\theta, X_n)]$, the behaviour of which can be compared to the asymptotic behaviour of (6).

The use of the ODE method for analysing learning algorithms like neural nets has originally been introduced by Benaïm [Benaïm, 1995]. An application to

the analysis of reinforcement learning algorithms has already

been considered in [Bertsekas and Tsitsiklis, 1996, Kushner and Yin, 1997], where convergence analysis of Q-Learning are presented. The point we want to emphasize in this article is that the ODE method can also be applied to study the learning rates of reinforcement learning algorithms like $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning.

The representation we adopt for this study is the following: the parameter vector θ to estimate is the optimal value function Q^* , X_n represents the observation at time n , and is defined as $X_n = (x_{n-1}, a_{n-1}, x_n)$, $H()$ is the update rule of $Q_{\mathcal{H}}$ -Learning in finite-horizon. Here $H(Q, (x, a, y))$ is set to the vector:

$$(x, a) \begin{pmatrix} 0 \\ \vdots \\ \begin{cases} r(x, a) + \max_b Q(y, b) - Q(x, a) & \text{if } x \in S_i, i < N \\ r(x, a) - Q(x, a) & \text{if } x \in S_N \end{cases} \\ \vdots \\ 0 \end{pmatrix}$$

Thus the two algorithms $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning can be described as

$$Q_{n+1} = Q_n + \frac{1}{n} \Gamma H(Q_n, X_{n+1}) \quad (7)$$

where $\Gamma = \Gamma^{Q_{\mathcal{H}}}$ or $\Gamma^{R_{\mathcal{H}}}$ is an adaptive gain matrix. For $\beta_n = 0$ and $\alpha_n(x, a) = \frac{1}{n}$, $\Gamma^{Q_{\mathcal{H}}} = \Gamma^{R_{\mathcal{H}}} = I$ which corresponds to the simplest version of $Q_{\mathcal{H}}$ -Learning. Therefore, within the ODE method, $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning can be considered as two discrete approximations with adaptive matrix-valued gains $\Gamma^{Q_{\mathcal{H}}}$ and $\Gamma^{R_{\mathcal{H}}}$ of the same differential equation:

$$\frac{dQ}{dt} = h(Q) \quad (8)$$

with $h(Q) = \lim_{n \rightarrow \infty} E_Q[H(Q, X_n)]$. $h(Q)$ can be calculated from the stationary distribution μ^Q of the Markov chain $\{X_n\}_n$ given a constant parameter Q : $h(Q) = \sum_X H(Q, X) \mu^Q(X)$.

To calculate the stationary distribution μ^Q we have to take into account the fact that for reinforcement learning algorithms, the input sequence $\{X_n\}_n$ is a Markov process controlled by the parameter vector Q_n itself [Benveniste et al., 1990]. For Markovian exploration

functions, where a_n depends probabilistically of x_n and Q_n , μ^Q is given by:

$$\forall X = (x, a, x'), \quad \mu^Q(X) = \mu_E^Q(x) P_{exp}^Q(a | x) P(x' | x, a)$$

where μ_E^Q is the stationary distribution of the Markov chain $\{x_n\}_n$ defined by $P(x' | x) = \sum_{a \in A_i} p_i(x' | x, a) P_{exp}^Q(a | x)$ for $x \in S_i$, and $P_{exp}^Q(a_n | x_n)$ is the selection probability of the exploration function. Since we added a uniform return from S_N to S_1 , we have $\forall x \in S_1, \mu_E^Q(x) = \frac{1}{N \cdot n_{S_1}}$. Iteratively, μ_E^Q can be computed on each state-space S_i as: $\forall x \in S_{i+1}, \mu_E^Q(x) = \sum_{z \in S_i} P(x | z) \mu_E^Q(z)$.

We easily check that Q^* is a stable attractive point of (8). First we can see that $h(Q^*) = 0$. Moreover, we can calculate the jacobian matrix of h on that point:

$$h_Q(Q^*) = \left(\frac{d}{dQ} h \right)(Q^*) = \quad (9)$$

$$(x, a) \begin{pmatrix} (x', a') \\ \vdots \\ \begin{cases} -\mu^*(x, a) & \text{if } (x, a) = (x', a') \\ p_i(x' | x, a) \mu^*(x, a) & \text{if } x \in S_i, i < N, \\ & x' \in S_{i+1}, a' = \pi^*(x') \\ 0 & \text{otherwise} \end{cases} \end{pmatrix}$$

with $\pi^*(x) = \operatorname{argmax}_a Q^*(x, a)$ and $\mu^*(x, a) = \mu_E^{Q^*}(x) P_{exp}^{Q^*}(a | x)$. The eigenvalues of $h_Q(Q^*)$ are equal to $-\mu^*(x, a)$. They are strictly negative with the simple assumptions that the Markov chain $\{x_n\}_n$ is recurrent at $Q = Q^*$, and that $\forall x, a P_{exp}^{Q^*}(a | x) > 0$.

Based on that material, we can now focus on the problem of the learning rate analysis, and its application to the comparison between $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning in finite-horizon MDPs.

3.3 Optimal matrix-valued learning rates

The use of a matrix-valued gain to guide and accelerate the convergence of a stochastic adaptive algorithm is a classic result of stochastic approximation theory [Benveniste et al., 1990, Kushner and Yin, 1997].

For the algorithm (7) the gain matrices Γ that maintain Q^* as a stable equilibrium of the new ODE

$$\frac{dQ}{dt} = \Gamma h(Q),$$

are characterized by $\forall \lambda$ eigenvalue of $\frac{1}{2}I + \Gamma \cdot h_Q(Q^*)$, $\operatorname{Re}(\lambda) < 0$. Among all these matrices, it is possible to prove [Benveniste et al., 1990,

Kushner and Yin, 1997] that the one that minimizes the asymptotic variance $\lim_{n \rightarrow \infty} \|Q_n - Q^*\|^2$ is defined by

$$\Gamma^* = -h_Q^{-1}(Q^*) \quad (10)$$

As we can see the knowledge of the target parameter Q^* is generally required and an adaptive matrix-valued gain Γ_n that converges toward Γ^* is often used. In our case, Γ^* can be calculated by inverting (9). We obtain the following upper triangular optimal matrix [Garcia and Ndiaye, 1998]:

$$\Gamma^* = \begin{pmatrix} & & (x', a') \\ & & \vdots \\ & & \frac{1}{\mu^*(x, a)} \text{ if } (x, a) = (x', a') \\ (x, a) & \dots & \begin{cases} \frac{P^{Q^*}(x' | x, a)}{\mu^*(x', a')} & \text{if } x \in S_i, x' \in S_j, \\ & i < j, a' = \pi^*(x') \\ 0 & \text{otherwise} \end{cases} \end{pmatrix}$$

where $P^{Q^*}(x' | x, a)$ is the probability of going from $x \in S_i$ to $x' \in S_j$, $1 \leq i < j \leq N$, in $j - i$ steps, by first executing the action a , and then by following the current policy $\pi^{Q^*}(x) = \operatorname{argmax}_a Q(x, a)$ for the last $j - i - 1$ steps.

3.4 Comparison between Γ^{Q_H} , Γ^{R_H} and Γ^*

For the two algorithms Q_H -Learning (2) and R_H -Learning (5) that we consider in this paper, the gains Γ^{Q_H} and Γ^{R_H} are adaptive gains that depend on n , but also on X_n and Q_n .

In order to be able to compare these matrix-valued gains with the optimal gain Γ^* , it is necessary to consider their asymptotic behaviour. If we assume classically that $\alpha_n(x, a) = \frac{\alpha_0}{N(x, a)}$ where $N(x, a)$ is the total number of times the pair (x, a) was visited at time n , and $\beta_n = \frac{\beta_0}{n}$, we show in [Garcia and Ndiaye, 1998] that Γ^{Q_H} and Γ^{R_H} converge respectively toward:

$$\Gamma_\infty^{Q_H} = \begin{pmatrix} \ddots & & 0 \\ & \frac{\alpha_0}{\mu^*(x, a)} & \\ 0 & & \ddots \end{pmatrix}$$

$$\Gamma_\infty^{R_H} = \begin{pmatrix} & & (x', a') \\ & & \vdots \\ & & \frac{\alpha_0}{\mu^*(x, a)} + (N-i+1)\beta_0 \text{ if } (x, a) = (x', a'), x \in S_i, a = \pi^*(x) \\ (x, a) & \dots & \begin{cases} \frac{\alpha_0}{\mu^*(x, a)} & \text{if } (x, a) = (x', a'), a \neq \pi^*(x) \\ (N-i+1)\beta_0 & \text{if } (x, a) \neq (x', a'), x \in S_i, a' = \pi^*(x') \\ 0 & \text{if } (x, a) \neq (x', a'), a' \neq \pi^*(x') \end{cases} \end{pmatrix}$$

A first remark about these matrix-valued gains is that the stability condition on Q^* implies that $\alpha_0 > \frac{1}{2}$. This explains some empirical results concerning R -Learning which reveal that higher initial values of α_0 are to be preferred to lower values [Mahadevan, 1996b].

It is now interesting to compare $\Gamma_\infty^{Q_H}$, $\Gamma_\infty^{R_H}$ and Γ^* . For $\alpha_0 = 1$ and a small β_0 , the three matrices have more or less the same diagonal values, which is a confirmation of the good choice $\alpha_n(x, a) = \frac{1}{N(x, a)}$, asymptotically equivalent to $\frac{1}{n\mu^*(x, a)}$.

Another important similarity between $\Gamma_\infty^{R_H}$ and Γ^* is about the structure of the matrices : both of them have exactly the same null columns.

4 Simulations

In order to experimentally compare Q_H -Learning, R_H -Learning and the Γ^* -Learning corresponding to (7) with $\Gamma = \Gamma^*$, we have developed a random finite-MDP generator. At each step i , a set S_i of n_S states and a set A_i of n_A actions are defined. Each transition from S_i to S_{i+1} is characterized by a set of n_A transition matrices $p_i(\cdot | \cdot, a)$ and n_A reward vectors $r_i(\cdot, a)$. The problem parameters are N , n_S and n_A . The reward values $r_i(s, a)$ and the probabilities $p_i(s' | s, a)$ are drawn in $[0, 1]$ from a random number generator, with the constraints $\sum_{s'} p_i(s' | s, a) = 1$.

For a given random MDP, we first calculate the exact finite-horizon optimal policy π^* with the classical N -step backward dynamic programming algorithm, using the p_i and r_i values. Then we calculate μ^* , P^{Q^*} , and finally the Γ^* optimal gain.

We evaluated the performance of the 3 algorithms Q_H -Learning, R_H -Learning and Γ^* -Learning on different random MDPs [Garcia and Ndiaye, 1998].

The learning parameters α_n et β_n were defined by

$$\alpha_n(x, a) = \frac{\alpha_0}{N(x, a)} \text{ and } \beta_n = \frac{\beta_0}{n/N},$$

where $N(x, a)$ is the number of times the action a has been chosen in the state x . We used $\alpha_0 = 1$ and $\beta_0 = 0.4$ for all simulations, with a semi-uniform exploration function ($\tau = 90\%$). These choices of learning rates were made to optimize the behaviour of Q_H -Learning and R_H -Learning on the set of problems we considered.

We chose $\rho_{\pi_n} = \frac{1}{N \cdot n_S} \sum_{x \in S_1} V_1^{\pi_n}(x)$ as a performance measure of the current policy π_n at iteration n , and

the variability of this policy π_n for different learning trajectories was taken into account by calculating the mean of the value ρ_{π_n} on M different runs (we took $M = 5$). More precisely we considered for each algorithm the two criteria $C_1 : \rho_{\pi_n}/\rho^*$ for $n = 500N$, and $C_2 : \rho_{\pi_n}/\rho^*$ for $n = 5000N$, where $\rho^* = \frac{1}{N \cdot n_S} \sum_{x \in S_1} V_1^*(x)$ is the optimal average gain of π^* .

The first surprising fact we noticed was that most of the time the Γ^*Q -Learning did not converge. An explanation we found is that for large sized problems, the initial gains $\frac{1}{\mu^*(x,a) \cdot n}$ of Γ^*Q -Learning are too large, and make the series $\{Q_n\}_n$ leaves its convergence domain. To fix that problem we decided to replace Γ^* by an adaptive matrix Γ_n^* asymptotically equivalent to Γ^* , where $\frac{n}{N(x,a)}$ is used instead of $\frac{1}{\mu^*(x,a)}$. The results we finally obtained showed that Γ_n^*Q -Learning is a bit better than $R_{\mathcal{H}}$ -Learning, and that both of them are always faster than $Q_{\mathcal{H}}$ -Learning, as illustrated in table 3 and figure 4.

$\rho_{\pi_n}/\rho^* \%$ N, n_S, n_A	$C_1 (n = 500N)$			$C_2 (n = 5000N)$		
	$Q_{\mathcal{H}}$	$R_{\mathcal{H}}$	Γ_n^*Q	$Q_{\mathcal{H}}$	$R_{\mathcal{H}}$	Γ_n^*Q
5,25,10	79.54	89.71	92.55	97.90	99.39	99.42
5,50,10	71.79	84.57	89.93	96.50	98.15	99.12
5,50,50	65.26	78.56	80.32	94.53	95.87	95.95
5,100,10	62.87	78.82	87.19	91.79	95.85	97.91
5,300,10	56.42	64.32	82.24	77.88	89.24	93.27
10,25,10	74.35	90.57	95.90	94.62	99.28	99.83
10,50,50	61.72	77.72	88.06	92.72	95.81	97.61
10,100,10	61.21	79.01	91.41	87.21	95.80	97.94
50,50,10	63.43	85.24	86.94	72.38	98.32	97.39
50,50,50	58.53	79.65	84.52	74.54	95.67	96.87

Figure 3: Relative evaluation of $Q_{\mathcal{H}}$ -Learning, $R_{\mathcal{H}}$ -Learning and Γ_n^*Q -Learning.

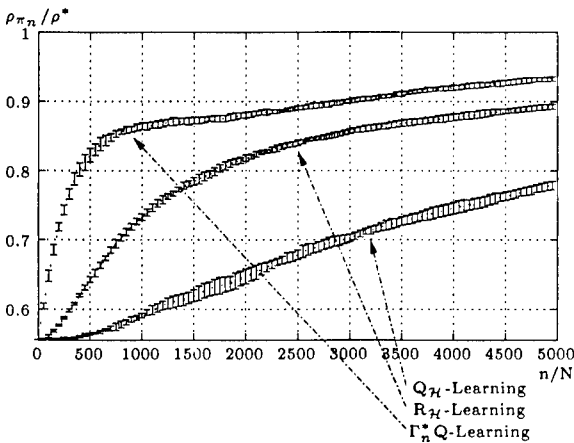


Figure 4: $n_A=10$, $n_S=300$, $N=5$ (5 runs).

5 Conclusion

The underlying goal of this article was to tackle the problem of using reinforcement learning algorithms in the framework of finite-horizon Markov Decision Processes. Two main results have been obtained.

First we proved the equivalence between the total reward criterion and the average-reward criterion in finite horizon. An interesting conclusion is that classical Q-Learning and R-Learning algorithms can be adapted to define $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning algorithms in finite horizon. Both of these algorithms converge experimentally toward the optimal V_i^* value functions, with a convergence proof for $Q_{\mathcal{H}}$ -Learning.

The other important result is about the comparison between the learning rates of $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning. It appears that $R_{\mathcal{H}}$ -Learning can be seen as a version of $Q_{\mathcal{H}}$ -Learning using matrix-valued step-sizes, where several components of the Q function are updated simultaneously. Furthermore, we showed that this stepsize matrix is structurally and numerically very close to the optimal gain matrix proposed by the ODE method, and that $R_{\mathcal{H}}$ -Learning performs very similarly to the learning algorithm corresponding to this optimal gain matrix. Consequently we argue in favor of using $R_{\mathcal{H}}$ -Learning when solving finite-horizon MDPs.

Different open questions still deserve to be considered. First we would like to know whether it is possible to derive from Γ_n^*Q -Learning an equivalent reinforcement learning algorithm where only one component is updated at each state transition, like it is the case for $R_{\mathcal{H}}$ -Learning in its initial formulation. For the moment, independently of the fact that it requires to know PQ^* and μ^* , Γ_n^*Q -Learning cannot be used in practice since it is too much slow.

Another question we are currently considering is to exploit the equivalent Q formulation of $R_{\mathcal{H}}$ -Learning for proving its convergence. Some recent theoretical results concerning the ODE method could be sufficient, like in [Benaïm et al., 1998].

Finally, we are trying to generalize our results concerning the convergence of $Q_{\mathcal{H}}$ -Learning and $R_{\mathcal{H}}$ -Learning to Q-Learning and R-Learning within the classical framework of stationary infinite MDPs.

Acknowledgements

We thank Michel Benaïm for fruitful discussions about the ODE method.

References

- [Benaïm, 1995] Benaïm, M. (1995). Convergence Theorem for Hybrid Learning Rules. *Neural Computation*, 7(1):19–25.
- [Benaïm, 1996] Benaïm, M. (1996). A Dynamical System Approach to Stochastic Approximations. *SIAM Control and Optimisation*, 34(2):437–472.
- [Benaïm et al., 1998] Benaïm, M., Fort, J. C., and Pages, G. (1998). Almost sure convergence of the one-dimensional kohonen algorithm. *Advances in Applied Probability*. To appear.
- [Benveniste et al., 1990] Benveniste, A., Metivier, M., and Priouret, P. (1990). *Adaptive Algorithms and Stochastic Approximation*. Springer-Verlag, Berlin, New York.
- [Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont (MA).
- [Garcia and Ndiaye, 1998] Garcia, F. and Ndiaye, S. (1998). Reinforcement Learning in finite-horizon: Optimality criteria and algorithms. Technical report, Department of Biometry and Artificial Intelligence, INRA, Toulouse, France.
- [Jaakkola et al., 1994] Jaakkola, T., Jordan, M., and Singh, S. (1994). On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, 6:1185–1201.
- [Kaelbling et al., 1996] Kaelbling, L., Littman, M., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285.
- [Kushner and Clark, 1978] Kushner, H. and Clark, D. (1978). *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, Berlin, Heidelberg, New York.
- [Kushner and Yin, 1997] Kushner, H. and Yin, G. (1997). *Stochastic Approximation Algorithms and Applications*. Springer.
- [Ljung, 1977] Ljung, L. (1977). Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control*, 22:551–575.
- [Mahadevan, 1996a] Mahadevan, S. (1996a). An Average-Reward Reinforcement Learning Algorithm for Computing Bias-Optimal Policies. In *AAAI National Conference*, volume 13.
- [Mahadevan, 1996b] Mahadevan, S. (1996b). Average Reward Reinforcement Learning: Foundations, Algorithms and Empirical Results. *Machine Learning*, 22:159–196.
- [Ok and Tadepalli, 1996] Ok, D. and Tadepalli, P. (1996). Auto-exploratory average reward reinforcement learning. In *AAAI National Conference*, volume 13.
- [Puterman, 1994] Puterman, M. (1994). *Markov Decision Processes*. John Wiley and Sons, New York.
- [Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.
- [Schwartz, 1993] Schwartz, A. (1993). A Reinforcement Learning Method for Maximizing Undiscounted Rewards. In *International Conference on Machine Learning*, volume 10.
- [Singh, 1994] Singh, S. (1994). Reinforcement Learning Algorithms for Average-Payoff Markovian Decision Processes. In *AAAI International Conference*, volume 12.
- [Tsitsiklis, 1994] Tsitsiklis, J. N. (1994). Asynchronous Stochastic Approximation and Q-Learning. *Machine Learning*, 16:185–202.
- [Watkins, 1989] Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England.
- [Watkins and Dayan, 1992] Watkins, C. and Dayan, P. (1992). Q-Learning. Technical Note. *Machine Learning*, 8(3):279–292.

Well-Behaved Borgs, Bolos, and Berserkers

Diana F. Gordon

Naval Research Laboratory, Code 5510
4555 Overlook Avenue, S.W.
Washington, DC 20375
gordon@aic.nrl.navy.mil

Abstract

How can we *guarantee* that our software and robotic agents will behave as we require, even after learning? Formal verification should play a key role but can be computationally expensive, particularly if *re*-verification follows each instance of learning. This is especially a problem if the agents need to make rapid decisions and learn quickly while online. Therefore, this paper presents novel methods for reducing the time complexity of re-verification subsequent to learning. The goal is agents that are predictable *and* can respond quickly to new situations.

1 INTRODUCTION

Software and robotic agents are becoming increasingly prevalent. Agent designers can furnish such agents with plans to perform desired tasks. Nevertheless, a designer cannot possibly foresee all circumstances that will be encountered by the agent. Therefore, in addition to supplying an agent with plans, it is essential to also enable the agent to learn and modify its plans to adapt to unforeseen circumstances. The introduction of learning, on the other hand, often makes the agent's behavior significantly harder to predict. Our objective is to develop methods that provide verifiable guarantees that the behavior of learning agents always remains within the bounds of specified constraints (called "properties"), even after learning. An example of a property is Asimov's First Law of Robotics (Asimov, 1942). This law, which has recently been studied by Weld and Etzioni (1994), states that a robot may not harm a human or allow a human to come to harm. Weld and Etzioni advocate a "call

to arms:' before we release autonomous agents into real-world environments, we need some credible and computationally tractable means of making them obey Asimov's First Law...how do we stop our artifacts from causing us harm in the process of obeying our orders?" Asimov's law can be operationalized into specific properties testable on a system, e.g., "Never delete another user's file." This paper addresses Weld and Etzioni's "call to arms" in the context of adaptive agents. It is a very important topic for real-world agents and is a dominant theme in science fiction, which is sometimes prescient. Examples include the Borgs (Star Trek, The New Generation), Bolos (Laumer, 1976), and Berserkers (Saberhagen, 1967) – fictional agents that demonstrate the dangerous behavior that can result from insufficient constraints.

We assume that an agent's plan has been initially verified offline. Then, the agent is fielded and has to adapt online. After adaptation via learning, the agent must rapidly *re*-verify its new plan to ensure this plan still satisfies required properties.¹ Re-verification must be as computationally efficient as possible because it is performed online, perhaps in a highly time-critical situation. There are numerous applications of this scenario, including software agents that can safely access information in confidential or proprietary environments while responding to rapidly changing access requirements, planetary rovers that quickly adapt to unforeseen planetary conditions but behave within critical mission constraints, and JAVA applets that can get smarter but not become destructive to our computing environments.

Typically, properties desired by a user are orthogonal to both the agent's planning goals and its learning

¹Current output is success/failure. Future work will consider using re-verification counterexamples to choose a better learning method when re-verification fails.

goals. For example, the agent may generate a plan with the objective of maximizing the agent's profit. Learning might have the goal of achieving the agent's plan more efficiently or modifying the plan to adapt to unforeseen events. The designer may have an additional constraint that the agent does not cheat in its dealings with other agents. Why doesn't the planner incorporate all properties into the plan? There are a number of possible reasons, e.g., not all properties may be known at the time the plan is developed, or security reasons.

Re-verification can be (from least to most time required): none, incremental, or complete. It is possible to avoid re-verification entirely if we restrict the agent to using only those learning methods determined a priori to be "safe" with respect to certain classes of properties in which we are interested. In other words, if a plan satisfies a property prior to learning, we want an a priori guarantee that the property will still be satisfied subsequent to learning. Note that this incurs *no* run-time cost. It is called "moving a tester into the generator" or "compiling constraints."

Unfortunately, the safety of some learning methods may be very difficult or maybe impossible to determine a priori. When a priori determination is too difficult, it is helpful to use incremental re-verification. Incremental methods save computational costs over re-verification from scratch by localizing re-verification and/or by reusing knowledge from the original verification. Furthermore, incremental methods may identify positive results that cannot be determined a priori. When an agent needs to learn, we suggest that the agent should consult the a priori results first. If no positive results exist, then incremental re-verification proceeds. The least desirable of the three alternatives is to do complete re-verification from scratch.

Gordon (1997a) begins to explore the extent to which we can prove a priori results that certain machine learning operators are, or are not, safe for certain classes of properties. The paper has positive a priori results for plan efficiency improvements via deletion of plan elements, as well as for plan refinement methods. Unfortunately, we have not yet obtained positive a priori results for popular machine learning operators such as abstraction (unless one is willing to accept an abstracted property) or generalization. Abstraction is a more global operator than generalization. Abstraction alters the language of a plan (e.g., by feature selection), whereas generalization alters the condition for a state-to-state transition within a plan. Both are extremely common operators in concept learning, but

are also very appropriate for plan modification.

This paper has two contributions beyond (Gordon, 1997a). First, the previous paper models agent plans using automata on infinite strings. This paper reaches a wider audience by using the more familiar automata on finite strings. Second, this paper addresses two, new questions: Are there situations in which an abstracted property is acceptable? If yes, we have positive a priori results for abstraction. Also, can we get positive results by using incremental re-verification rather than a priori? Initial, positive answers to these questions are presented here.

The remainder of this paper is organized as follows. Section 2 presents an illustrative example that is used throughout the paper.² Section 3 contains background material and definitions on automaton plans, temporal logic properties, and "safe" learning. The formal definitions provide a precise foundation for understanding the incremental re-verification methods presented later. Section 4 lists situations in which property abstraction is acceptable. Sections 4 and 5 present novel (and as far as we are aware, the only) methods for incremental re-verification of abstraction and generalization, respectively, on automata. Finally, time complexity comparisons between incremental and complete re-verification are provided.

2 ILLUSTRATIVE EXAMPLE

This section provides an example to illustrate some of the main ideas of the paper. Although the plan in this example is very small, it is important to point out that existing automata-based verification methods currently handle huge, industrial-sized problems (e.g., see Kurshan, 1994). Our goal is to improve the time complexity of verification over current methods when learning occurs.

In our example, hundreds of tiny, micro air vehicles (MAVs) are required to perform a task within a region. The MAVs are divided into two groups called "swarm A" and "swarm B." One constraint, or property, is that only one MAV may enter the region at a time – because multiple MAVs entering simultaneously would increase the risk of detection. Each swarm has a separate FIFO queue of MAVs. MAVs enter the queue when they return from their last task. A second constraint is that some (at least one) MAVs from each swarm eventually

²Examples in this paper have been implemented using Kurshan's COSPAN verification system. COSPAN is an AT&T verification tool, which is described in Kurshan (1994).

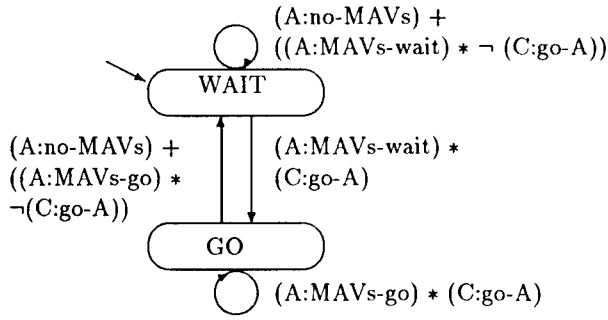


Figure 1: Plan A

enter the region. One distinguished MAV, C, acts as a task coordinator. C selects which swarm, A or B, may send in an MAV next.³

Plans for swarm A and task controller C are shown in Figures 1 and 2. The plan for swarm B is not shown in the figure, but it is identical to the plan for A except all instances of “A” are replaced by “B.” Each of these plans is a finite-state automaton, i.e., a graph with states (the vertices) and allowable state-to-state transitions (the directed edges between vertices). The transition conditions (i.e., the logical expressions labeling the edges) describe the set of actions that enable a state transition to occur. The possible actions A can take from a state are (A:no-MAVs), (A:MAVs-wait), or (A:MAVs-go). The first action means the queue is empty, the second that the queue is not empty but the MAVs in the queue must wait, and the third that the first MAV in the queue enters the region. Likewise for B. The possible actions C can take from a state are (C:go-A) or (C:go-B). The first action means controller C allows swarm A to send one MAV into the region, the second means C allows B to send one MAV into the region.

Swarms A and B are single agents, i.e., although individual MAVs may each have their own plan, such as queuing within a swarm, for simplicity we ignore that level of detail. We can form a multiagent plan by taking a “product” (see Section 3.1) of the plans for A, B, and C. This product synchronizes the behavior of A, B, and C in a coordinated fashion. At every discrete time step, every agent (A, B, C) is at one state in its plan, and it selects its next action. The action of one agent (e.g., A) becomes an input to the other agents’ plans (e.g., B and C). If the joint actions chosen by all three agents satisfy the transition conditions of a plan from the current state to some next state, then that

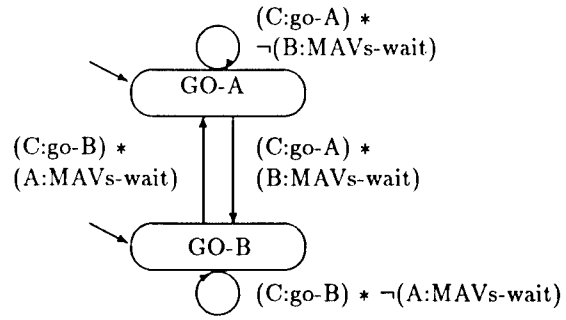


Figure 2: Plan C

transition may be made. For example, if the agents jointly take the actions (A:MAVs-wait) and (B:MAVs-wait) and (C:go-A), then the multiagent plan can transition from the global, joint state (WAIT, WAIT, GO-A) to the joint state (GO, WAIT, GO-B) represented by triples of states in the automata for agents A, B, and C.

Given the full, multiagent plan, verification now consists of asking the question: Does this plan satisfy the two required properties, i.e., some MAVs from each swarm enter the region, but only one MAV enters the region at a time? Assuming our initial plan in Figures 1 and 2 satisfies these properties, we next ask whether the properties are still satisfied subsequent to learning. The latter question is the topic of this paper.

An example of learning is the following. Suppose coordinator C discovers that the B swarm has left the region. One way agent C can adapt to incorporate this new knowledge is by deleting the action (C:go-B) from its action repertoire. This is a form of abstraction. There are alternative modifications agent C can do, but the selection between these alternatives is a learning issue, which we do not address here. What we do address here are the implications of this choice, in particular, which learning methods are safe, i.e., preserve the properties.

3 PLANS, PROPERTIES, AND “SAFE” LEARNING

3.1 AUTOMATON PLANS

This subsection, which is based on Kurshan (1994), briefly summarizes the basics of the automata used to model plans. Figures 1 and 2 illustrate the definitions. Essentially, an automaton is a graph with vertices corresponding to states and directed edges corresponding to state-to-state transitions. The terms “ver-

³This example is a variant of the traffic controller in Kurshan (1994).

tex” and “state” are used interchangeably throughout the paper. For an automaton representing an agent’s plan, vertices represent the internal state of the agent and/or the state of its external environment. State-to-state transitions have associated transition conditions, which are the conditions under which the transition may be made. An agent action that satisfies a transition condition enables that transition to be made. We assume finite-state automata, i.e., the set of states is finite, and that the transition conditions are elements of a Boolean algebra. Therefore, we briefly diverge to summarize the basics of Boolean algebras.

A Boolean algebra \mathcal{K} is a set with distinguished elements 0 and 1, closed under the Boolean operations $*$ (logical “and”), $+$ (logical “or”), and \neg (logical negation), and satisfying the standard properties (Kurshan, 1994).

The Boolean algebras are assumed to be finite. There is a partial order among the elements, \preceq , which is defined as $x \preceq y$ if and only if $x * y = x$. The elements 0 and 1 are defined as $\forall x \in \mathcal{K}, 0 \preceq x$ and $\forall x \in \mathcal{K}, x \preceq 1$. The *atoms* of \mathcal{K} , $\Gamma(\mathcal{K})$, are the nonzero elements of \mathcal{K} minimal with respect to \preceq . For two different atoms x and y within the *same* Boolean algebra, $x * y = 0$. For Figures 1 and 2, agents A, B, and C each have their own Boolean algebra with its atoms. The atoms of A’s Boolean algebra are the actions (A:no-MAVs), (A:MAVs-wait), and (A:MAVs-go); the atoms of B’s algebra are (B:no-MAVs), (B:MAVs-wait), and (B:MAVs-go); the atoms of C’s algebra are (C:go-A) and (C:go-B).

A Boolean algebra \mathcal{K}' is a *subalgebra* of \mathcal{K} if \mathcal{K}' is a non-empty subset of \mathcal{K} that is closed under the operations $*$, $+$, and \neg , and also has the distinguished elements 0, 1. Let $\mathcal{K} = \prod \mathcal{K}_i$, i.e., \mathcal{K} is the product algebra of the \mathcal{K}_i . In this case the \mathcal{K}_i are subalgebras of \mathcal{K} . An atom of the product algebra is the product of the atoms of the subalgebras. For example, if a_1, \dots, a_n are atoms of subalgebras $\mathcal{K}_1, \dots, \mathcal{K}_n$, respectively, then $a_1 * \dots * a_n$ is an atom of \mathcal{K} .

In Figure 1, the Boolean algebra \mathcal{A} used by agent A is the smallest one containing the atoms of A’s algebra. It contains all Boolean elements formed from A’s atoms using the Boolean operators $*$, $+$, and \neg , including 0 and 1. These same definitions hold for B and C’s algebras \mathcal{B} and \mathcal{C} . One atom of the product algebra \mathcal{ABC} is (A:no-MAVs) $*$ (B:no-MAVs) $*$ (C:go-A). This is the form of actions taken by the three agents in the multiagent plan. Algebras \mathcal{A} , \mathcal{B} , and \mathcal{C} are subalgebras of the product algebra \mathcal{ABC} . Finally, \mathcal{ABC} is the

Boolean algebra for the transition conditions in the multiagent plan.

Let us return now to automata. This paper focuses on automata that model agents with finite lifetimes (represented as a finite string, or sequence of actions). An example is an agent that is created specially to execute a plan and is destroyed immediately afterwards. In particular, we focus on *processes*. Processes are automata, but they are the *dual* of our usual notion of an automaton, which accepts any string beginning in an initial state and ending in a final state (Hopcroft & Ullman, 1979). Instead, processes accept any string beginning in an initial state and ending in a non-final state.⁴ A string is a sequence of actions (atoms). Therefore, by specifying the set of final states, we can infer the set of action sequences not permitted by the plan. It consists of those strings ending in a final state. All other action sequences that begin in an initial state are permitted by the plan. Processes are used here to be consistent with the automata theoretic verification literature.

Formally, a process is a three-tuple $S = (M_{\mathcal{K}}(S), I(S), F(S))$ where \mathcal{K} is the Boolean algebra corresponding to S . $M_{\mathcal{K}}(S) : V(S) \times V(S) \rightarrow \mathcal{K}$ is the matrix of transition conditions, which are elements of \mathcal{K} , $V(S)$ is the set of vertices of S , $I(S) \subseteq V(S)$ are the initial states, and $F(S) \subseteq V(S)$ are the final states. Also, $E(S) = \{e \in V(S) \times V(S) \mid M_{\mathcal{K}}(e) \neq 0\}$ is the set of directed edges connecting pairs of vertices of S , and $M_{\mathcal{K}}(e)$ is the transition condition of $M_{\mathcal{K}}(S)$ corresponding to edge e . Note that we omit edges labeled “0.” By our definition, an edge whose transition condition is 0 does not exist. We can alternatively denote $M_{\mathcal{K}}(e)$ as $M_{\mathcal{K}}(v_i, v_{i+1})$ for the transition condition corresponding to the edge going from vertex v_i to vertex v_{i+1} . For example, in Figure 1, $M_{\mathcal{K}}$ (WAIT, GO) is (A: MAVs-wait) $*$ (C: go-A).

Figures 1 and 2 illustrate the process definitions. There are process plans for two agents: swarm A and task coordinator C. Recall that agent B is identical to A but with “A” replaced by “B.” An incoming arrow to a state, not from any other state, signifies that this is an initial state. Recall that the output actions of process A are its atoms, and likewise for processes B and C. The transition conditions are the labels on the edges. We assume for process $X = A, B$, or C , $F(X) = \emptyset$, i.e., there are no final states. Therefore every finite string of actions that starts in an initial

⁴For the case of deterministic and complete transition conditions, reversing the acceptance condition will complement the language.

state and satisfies the transition conditions is acceptable behavior for the plan.

A multiagent plan is formed from single agent plans by taking the *tensor product* of the processes corresponding to the individual plans. Essentially, this is done by taking the Cartesian product of the vertices and the intersection of the transition conditions. For details see Kurshan (1994). The product process models a set of synchronous processes. The Boolean algebra corresponding to the product process is the product algebra. For Figures 1 and 2, to formulate the process S modeling the entire multiagent plan, we take the tensor product $S = A \otimes B \otimes C$ of the three processes. For this tensor product, $I(S) = \{ (\text{WAIT}, \text{WAIT}, \text{GO-A}), (\text{WAIT}, \text{WAIT}, \text{GO-B}) \}$, and $F(S) = \emptyset$. The tensor product process is not shown in a figure because it's quite large.

Formally, a *string* \mathbf{x} is a finite-dimensional vector, $(x_0, \dots, x_n) \in \Gamma(\mathcal{K})^+$, i.e., a string is a sequence of one or more actions. A *run* \mathbf{v} of string \mathbf{x} is a sequence (v_0, \dots, v_{n+1}) of vertices such that $\forall i, 0 \leq i \leq n$, $x_i * M_{\mathcal{K}}(v_i, v_{i+1}) \neq 0$, i.e., $x_i \leq M_{\mathcal{K}}(v_i, v_{i+1})$ because the x_i are atoms.

The *language* of S is $\mathcal{L}(S) = \{ \mathbf{x} \in \Gamma(\mathcal{K})^+ \mid \mathbf{x} \text{ has a run in } M_{\mathcal{K}}(S) \text{ from } I(S) \text{ to } V(S) \setminus F(S) \}$. Such a run is *accepting*. The language of a plan is the set of all action sequences (i.e., strings) allowed by the plan.

An example string in the language of process S , the multiagent process that is the product of A , B , and C , is $((A:\text{MAVs-wait}) * (B:\text{MAVs-wait}) * (C:\text{go-A})), ((A:\text{MAVs-go}) * (B:\text{MAVs-wait}) * (C:\text{go-B})), ((A:\text{MAVs-wait}) * (B:\text{MAVs-go}) * (C:\text{go-B})), ((A:\text{MAVs-wait}) * (B:\text{MAVs-go}) * (C:\text{go-A}))$. This is a sequence of atoms of S . An accepting run of this string is $((\text{WAIT}, \text{WAIT}, \text{GO-A}), (\text{GO}, \text{WAIT}, \text{GO-B}), (\text{WAIT}, \text{GO}, \text{GO-B}), (\text{WAIT}, \text{GO}, \text{GO-A}), (\text{GO}, \text{WAIT}, \text{GO-A}))$. Because $F(S) = \emptyset$, all runs beginning in an initial state are accepting runs and they form the elements of the language of S .

3.2 TEMPORAL LOGIC PROPERTIES

We assume properties are expressed in temporal logic. For formal versions of the definitions here, see Manna and Pnueli (1991). Linear time is assumed here. In other words, time proceeds linearly and we do not consider simultaneous possible futures. The type of verification used in this paper is "model checking." In other words, verification tests whether $S \models P$ for plan S and property P , i.e., whether plan S "models," or satisfies, property P .

For consistency with the temporal logic literature, we define a *computational state* (*c-state*) as the action chosen from each process state. Then a computation is a finite sequence of temporally ordered computational states, i.e., a string. To distinguish the two types of states, we will refer to a process state as a *p-state*.

P is a property true (false) for a process S , i.e., $S \models P$ ($S \not\models P$), if and only if it is true for every string in the language $\mathcal{L}(S)$ (false for some string in $\mathcal{L}(S)$). The notation $\mathbf{x} \models P$ ($\mathbf{x} \not\models P$) means string \mathbf{x} satisfies (does not satisfy) property P , i.e., the property holds (does not hold) for \mathbf{x} . Before defining what it means for properties to be true (i.e., hold) for a string, we first define what it means for a formula that is Boolean expression to be true at a c-state. A *c-state formula* p is true (false) at c-state x_i , i.e., $x_i \models p$ ($x_i \not\models p$) if and only if $x_i \leq p$ ($x_i \not\leq p$), i.e., $x_i * p \neq 0$ ($= 0$) because p is a Boolean expression with no variables on the same Boolean algebra used by process S , and x_i is an atom of that algebra. For example, $(A:\text{MAVs-wait}) \models ((A:\text{MAVs-wait}) + (A:\text{no-MAVs}))$ for c-state $(A:\text{MAVs-wait})$ and c-state formula $((A:\text{MAVs-wait}) + (A:\text{no-MAVs}))$.

A c-state formula p is true/false in particular c-states of a string. Property P is defined in terms of p , and is true/false of an entire string, i.e., $\mathbf{x} \models P$ or $\mathbf{x} \not\models P$ for string \mathbf{x} . We now define two property classes that are among those most frequently encountered in the verification literature for finite strings. Assume $\mathbf{x} = (x_0, \dots, x_n)$ is a string of process S . For c-state formula p and plan S , define Sometimes property $P = \Diamond p$ ("Sometimes p ") as a property that is true for string \mathbf{x} if only if p is true in at least one c-state x_i of \mathbf{x} , where $0 \leq i \leq n$. An Invariance property $P = \Box p$ ("Invariant p ") is a property true for string \mathbf{x} if and only if p is true in every c-state x_i of \mathbf{x} .

Continuing with the MAVs example, a desirable Invariance property P_I states that "only one MAV enters the region at a time." This can be expressed in temporal logic as $P_I = \Box(\neg((A:\text{MAVs-go}) * (B:\text{MAVs-go})))$. A desirable Sometimes property P_S states that "Sometimes MAVs from swarm A enter the region." In logic this property is expressed as $P_S = \Diamond(A:\text{MAVs-go})$. P_I , but not P_S , holds for the multiagent plan S .

3.3 "SAFE" LEARNING

This paper is concerned with "safe" machine learning methods (*SMLs*), i.e., machine learning operators that preserve properties, also called "correctness preserving mappings." For plan S and property P , suppose

verification has succeeded prior to learning, i.e., $\forall \mathbf{x}, \mathbf{x} \in \mathcal{L}(S)$ implies $\mathbf{x} \models P$ (i.e., $S \models P$). Then according to Gordon (1997a), a machine learning operator $ml(S)$ is an SML if and only if verification succeeds after learning, i.e., $\forall \mathbf{x}, \mathbf{x} \in \mathcal{L}(ml(S))$ implies $\mathbf{x} \models ml(P)$. Note that a machine learning operator may also affect the property P , which could be undesirable. Therefore, being an SML is not always sufficient. Additional requirements on learning – in particular, abstraction, are discussed next.

4 BOOLEAN ALGEBRA ABSTRACTION

Kurshan (1994) presents methods for improving the efficiency of automata-based verification, but does not consider the possibility of automata, such as agents, that can learn. By applying some of the results of Kurshan (1994) in a novel way, Gordon (1997a; 1997b) shows that when agents learn using certain abstractions, the abstractions are a priori guaranteed to be SMLs for *all* property classes – but *only* if abstraction is performed to both the plan and property.⁵ Therefore, this section identifies situations in which it is acceptable to apply an SML abstraction to a property.

The SML abstractions include very useful ones, such as partitioning the Boolean algebra atoms e.g., using constructive induction, and projection, which is a form of feature selection (or, more properly, action deletion). Although the methods described in this section apply to any of these abstractions, for illustration we focus only on projection, which is a mapping from a Boolean algebra to a subalgebra. For a formal definition of projection, see Kurshan (1994). Here, we continue with the MAVs example.

Suppose all the MAVs in the B swarm leave the region. To incorporate this knowledge, Boolean algebra projection, a type of abstraction, projects the product algebra \mathcal{ABC} onto subalgebra \mathcal{AC} . Projection $proj_{\mathcal{AC}} : \mathcal{ABC} \rightarrow \mathcal{AC}$ is defined as $proj_{\mathcal{AC}}(a * b * c) = a * c$ for atoms $a \in \Gamma(\mathcal{A})$, $b \in \Gamma(\mathcal{B})$ and $c \in \Gamma(\mathcal{C})$, and is extended linearly to the full algebra. For example, $proj_{\mathcal{AC}}((A: \text{MAVs-wait}) * (B: \text{MAVs-wait}) * (C: \text{go-A})) = (A: \text{MAVs-wait}) * (C: \text{go-A})$. In addition to removing entire subalgebras, it is also possible to remove atoms from within a subalgebra.

Projection $proj_{\mathcal{AC}}$ removes all references to swarm B from the multiagent plan S. This projection reduces

B's plan to the trivial plan which allows B to do anything. We assume that when the agent applies a projection to the plan, it has justification to do so – because the purpose of abstraction is to modify the plan. Modification of the property, on the other hand, may be a side effect required for an a priori guarantee that the abstraction is an SML. Applying $proj_{\mathcal{AC}}$ to the Invariance property P_I , which states that “only one MAV may enter the region at a time,” results in a property which accepts *any* multiagent plan of agents A, B, and C. When applied to both plan and property, $proj_{\mathcal{AC}}$ is an SML. Nevertheless, if the B swarm returns to the region and is restored into the multiagent plan, then this new property which allows the agents to do anything could have disastrous, unintended (by the user) consequences.

This example illustrates our dilemma: If we abstract the property along with the plan, the abstraction will be guaranteed a priori to be an SML. However, by abstracting the property, we risk violating the user's original intentions. *When is it ok to abstract a property?* There are at least three cases when it is permissible:

- (1) When the abstraction is *property invariant*.

Applying the projection $proj_{\mathcal{AC}}$ to the Sometimes property P_S , which states that “Sometimes MAVs from swarm A enter the region,” leaves P_S invariant, i.e., $proj_{\mathcal{AC}}(P_S) = P_S$. Therefore the abstraction is property invariant. The intuition is that the behavior of agent B is irrelevant when testing this property.

In general, to determine whether property invariance holds, an agent must apply abstraction to each property P and then check whether P remains unaltered by abstraction. This simple syntactic check is a form of incremental re-verification because it is localized to a test on the property alone. The check has a worst case time complexity of $O(|P|)$ for any property P . This is lower than the worst case time complexity of complete re-verification from scratch (following abstraction), which is $O(|\Gamma(\mathcal{K})| * |P|)$ for Invariance and Sometimes properties, where $|\Gamma(\mathcal{K})|$ is the number of atoms in the plan (Lichtenstein & Pnueli, 1984). Furthermore, if the agent will only accept property invariant abstractions, then the cost of plan abstraction can be avoided when this incremental check fails.

- (2) When the abstraction is *property irrelevant*.

An example is when the agents discover, or are told about, a *permanent* change that henceforth renders one or more items (e.g., an agent or action) irrelevant. The term “permanent” in this context means a change

⁵This result applies to agents with finite or infinite lifetimes.

whose effects are sustained at least until the last agent has terminated. Because the change is permanent, we can be assured that no problems are caused by applying an SML abstraction to the properties.

Consider an example in which a swarm agent becomes irrelevant. Suppose the lives of all MAVs in the B swarm have terminated, e.g., they become permanently inoperative, but we wish to continue with the multiagent plan because the other agents survived. Then the application of $proj_{AC}$ to the property P_I has no significant effect – because P_I is no longer needed.

(3) When the abstraction is *property reversible*.

Suppose the agents determine that one or more items are not relevant to the objectives of their multiagent plan, but this is a *temporary* change in condition, i.e., the items may become relevant again. For example, agents may disappear to attend to different tasks then possibly return, and actions may become temporarily disabled due to mechanical failures. Items irrelevant to the multiagent objectives could be removed from the multiagent plan and also from the properties. Under these circumstances, we want the abstraction to be property reversible. An abstraction is property reversible if the pre-abstraction property can be restored, e.g., by saving it. This way we can retest the original property after undoing the effects of abstraction.

We only want our agents to perform property irrelevant and property reversible abstractions when abstraction is restricted to removing irrelevant items. If agents are not told relevance, they may need to perform relevance determination, perhaps using methods such as those of Subramanian (1988). Other research related to the ideas in this section includes feature selection (see <http://ai.iit.nrc.ca/bibliographies/feature-selection.html>), and plan abstraction (Knoblock, 1990).

5 GENERALIZATION

Although we have been unable to obtain positive a priori results for generalization, this section presents a novel method for incremental re-verification after generalization. Efficiency is gained by tailoring incremental re-verification methods to specific property classes. Because there are only about a dozen property classes commonly used in practice (Kurshan, 1994), this seems reasonable to do. The re-verification method presented in this section is specific to Invariance properties. A method for Sometimes properties

may be found in Gordon (1997b). Methods for other property classes are currently being investigated.

Generalization differs from abstraction in that you are not changing the entire Boolean algebra (e.g., taking a subalgebra) but instead you are increasing the generality of a transition condition labeling one or more edges (for simplicity, here we consider one). Generalization is done when the agent discovers that the transition can/should be taken under a larger set of circumstances. It is only done to the plan. In the context of a process, generalization raises the level of a particular p-state-to-p-state transition condition in the partial order \preceq , whereas specialization lowers it, e.g., as in Mitchell's Version Spaces (Mitchell, 1978).

Formally, we define generalization of the condition along edge (v, w) as follows. Generalization operator $ml_{gen} : S \rightarrow S'$, where both S and S' use Boolean algebra \mathcal{K} , is defined as $ml_{gen} : M_{\mathcal{K}}(S) \rightarrow M_{\mathcal{K}}(S')$, where $ml_{gen}(M_{\mathcal{K}}(v, w)) = M_{\mathcal{K}}(v, w) + z$, for some $z \in \mathcal{K}$.⁶

An example of generalization is the following. The transition condition associated with the edge ((WAIT, WAIT, GO-A), (GO, WAIT, GO-B)) in the multiagent plan S is $(A:MAVs-wait) * (B:MAVs-wait) * (C:go-A)$. This could be generalized to $((A:MAVs-wait) * (B:MAVs-wait) * (C:go-A)) + ((A:MAVs-wait) * \neg(B:MAVs-wait) * (C:go-A))$, i.e., $(A:MAVs-wait) * (C:go-A)$ for new plan S' .

To illustrate our incremental approach, recall S satisfies the Invariance property P_I which states that “only one MAV enters the region at a time,” i.e., $\Box (\neg ((A:MAVs-go) * (B:MAVs-go)))$. We could check this property against the entire, new plan S' , but a preferable alternative is to simply check it against the new addition to the transition condition, namely, is P_I satisfied by $(A:MAVs-wait) * \neg(B:MAVs-wait) * (C:go-A)$? In fact it is, because $(A:MAVs-go)$ is not true, and that is all we need to know to be sure that the ml_{gen} just applied is an SML. We can now formalize this.

Let us consider the Invariance property $P = \Box p$ for c-state formula p . Let y be the existing transition condition for edge (v, w) in plan S , i.e., $M_{\mathcal{K}}(v, w) = y$. We previously defined what it means for a c-state formula p to be true at a c-state, but it is also useful to define what it means for a c-state formula to be true of a transition condition. Let $\Gamma(\mathcal{K})_y = \{a \mid a \in \Gamma(\mathcal{K}) \text{ and } a \preceq y\}$. A c-state formula p is defined to be true of a transition condition y , i.e., $y \models p$, if and only if $\forall a \in \Gamma(\mathcal{K})_y, a \preceq p$.

⁶ S' differs from S only by the results of ml_{gen} .

Assume every string \mathbf{x} in $\mathcal{L}(S)$ satisfies Invariance property P , so for each \mathbf{x} , p is true of every atom in \mathbf{x} . This implies $y \models p$.⁷ Now we generalize the edge (v, w) to form S' via $ml_{gen}(M_K(v, w)) = y + z$. This operator ml_{gen} is an SML with respect to Invariance property P if and only if $S' \models P$, which is true if and only if $z \models p$. The reason for this is that we know S satisfies P from our original verification, and therefore p is true for all atoms in all strings in $\mathcal{L}(S)$. The only new atoms in $\mathcal{L}(S')$ but not in $\mathcal{L}(S)$ are in $\Gamma(K)_z$. Therefore, if $z \models p$, then p is true for all atoms in $\mathcal{L}(S')$, which implies every string in $\mathcal{L}(S')$ satisfies P , i.e., $S' \models P$. Therefore, re-verification need only test whether $z \models p$, i.e., $\forall a \in \Gamma(K)_z, a \preceq p$. (We assume transition conditions are represented extensionally, i.e., as the unique sum of atoms equivalent to the Boolean expression.) If $z \not\models p$, $S' \not\models P$.⁸ This test is *incremental* because it is localized to just checking whether the property holds of the newly added atoms in z , rather than all atoms in $\mathcal{L}(S')$.

For example, suppose a, b, c, d , and e are atoms, and the transition condition y between v and w equals a . Let (a, b, b, d) be an accepting string of S that includes v and w as the first two vertices in its accepting run. The property is $P = \Box \neg e$. Assume the fact that this string satisfies $\neg e$ was proved in the original verification. Suppose ml_{gen} generalizes $M_K(v, w)$ from a to $(a + c)$, which adds the string (c, b, b, d) to $\mathcal{L}(S')$. Then rather than test whether the elements of $\{a, b, c, d\}$ are $\preceq \neg e$, all we really need to test is whether $c \preceq \neg e$ – because c is the only newly added atom.

By storing and reusing knowledge from previous verification(s), we can increase the efficiency of this test. Suppose some atoms a such that $a \preceq z$ were tested for $a \preceq p$ during previous verification(s), and the outcomes of these tests were stored. Then lookup will suffice, and the only atoms in $\Gamma(K)_z$ that need to be tested against p during the current re-verification are those not previously tested.

What cost benefit(s) does incremental re-verification have over complete re-verification from scratch? Verification, or complete re-verification from scratch, in the worst case has time complexity $O(|\Gamma(K)| * |p|)$ for Invariance properties, where $|\Gamma(K)|$ is the total number of atoms, and $|p|$ is the length of the c-state formula p (Lichtenstein & Pnueli, 1984). This is because the

c-state formula may have to be tested in every unique c-state, which is an atom. $|\Gamma(K)|$ is exponential in the number of single agent plans forming a multiagent plan. In the worst case, incremental re-verification has the same time complexity, but this would be a very bizarre situation indeed. It would require that no atoms were tested against the property in the original verification (which could occur if $\mathcal{L}(S)$ were empty), and *all* atoms are added to the transition condition during generalization, i.e., $\forall a \in \Gamma(K), a \preceq z$.

Let us consider a more realistic comparison. The worst case time complexity for complete re-verification assumes all c-states are reachable from some initial p-state. This may not be true, e.g., the number of initial p-states might be very small. Re-verification is required to determine $\forall \mathbf{x} \in \mathcal{L}(S')$ whether $\mathbf{x} \models P$. At the very least, complete re-verification of an Invariance property $P = \Box p$ must test whether $x_i \models p \forall x_i$ in \mathbf{x} , $\forall \mathbf{x} \in \mathcal{L}(S')$. The complexity of this test is $C_{complete} = O(|\Gamma(K)_{\mathcal{L}(S')}| * |p|)$, where $|\Gamma(K)_{\mathcal{L}(S')}|$ is the number of unique atoms in all strings $\mathbf{x} \in \mathcal{L}(S')$.

A more realistic cost estimate for incremental re-verification is $C_{increm} = O(|\Gamma(K)_{s(z)}| + (|\Gamma(K)_{ns(z)}| * |p|))$, where $\Gamma(K)_{s(z)}$ ($\Gamma(K)_{ns(z)}$) contains atoms whose results are (are not) previously stored. The first addend is the cost of lookup of results from previous verification(s), and the second addend is the cost added by testing the atoms that were not previously tested. Whenever generalization is reasonably conservative, i.e., $|\Gamma(K)_z| \ll |\Gamma(K)_{\mathcal{L}(S')}|$, incremental can provide considerable savings over complete re-verification!

6 DISCUSSION

Here we have addressed the question of how agents can adapt (learn) safely, i.e., by preserving critical properties, and how they can do this in a time-efficient manner. We extended the work of Gordon (1997a) to obtain positive results for two popular machine learning methods: abstraction and generalization. For abstraction to be a priori safe (property-preserving), the property must also be abstracted. This paper enumerates situations in which it is permissible to abstract the property. Furthermore, novel incremental re-verification methods are presented for abstraction and generalization. These methods have the potential to provide large computational savings over complete re-verification from scratch. With our methods (including a priori), agents can use abstraction and generalization to adapt to novel situations, and can do so with quick checks that ensure the reliability of their

⁷This statement is based on our assumption that (v, w) is part of an accepting run for at least one $\mathbf{x} \in \mathcal{L}(S)$. This assumption motivates re-verification.

⁸That is, unless (v, w) is not part of any accepting run – but then the test is unnecessary.

behavior.

There is a small amount of prior research on incremental re-verification. Reps and Teitelbaum (1989) developed a verifier for users to check their code while writing in traditional programming languages, such as PL/I. Their verifier can incrementally re-check software after edits using Hoare-style proofs. However, unlike our re-verification methods, these proofs require some interaction with the user. Sokolsky and Smolka (1994) have an incremental method for verifying added or deleted state transitions in an automaton-like representation. However they do not address generalization or abstraction. Finally, Weld and Etzioni (1994) have a method to incrementally test an agent's plan to decide whether to add new actions to the plan. There are certain similarities between our work and that of Weld and Etzioni. They add actions to a plan only when their effects do not violate *dont-disturb* properties, which are a type of Invariance property. Our generalization also adds actions to a plan. Furthermore, both approaches localize verification. The main differences are that unlike Weld and Etzioni, we: (1) use a formal foundation based on the verification literature, in particular, model-checking and automata, (2) assume the existence of prior verification knowledge and use this knowledge to streamline re-verification, (3) use reactive rather than necessarily goal-oriented plans, and (4) address abstraction.

One aspect of Weld and Etzioni (1994) that was purposely not addressed here is that of how to select which method to use in repairing a plan. This is a rich issue for future research, and could draw on cost-effective methods such as those of Joslin and Pollack (1994). Rather than repair, this paper focuses on re-verification. We are unaware of any methods besides ours for incrementally re-verifying abstraction or generalization in automata. Much more work remains to be done on the important topic of incremental re-verification – especially for adaptive agents.

Acknowledgments

Thanks to Bill Spears and Sampath Kannan for useful inputs on this paper. Bill suggested the alliterative title. This research was sponsored by the Office of Naval Research N001498WX20296.

References

- Asimov, I. (1942). Runaround. In *Astounding Science Fiction*.
- Gordon, D. (1997a). Asimovian adaptive agents. NCARAI Technical Report 97-016. Also submitted to *Machine Learning*.
- Gordon, D. (1997b). Machine learning and finite-lifetime agents: Some preliminary results. NCARAI Technical Report 97-017.
- Hopcroft, J. & Ullman J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Menlo Park: Addison-Wesley.
- Joslin, D. & Pollack, M. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. *Proceedings of AAAI94* (pp. 1004-1009). AAAI Press.
- Knoblock, C. (1990). A theory of abstraction for hierarchical planning. In D. P. Benjamin (Ed.), *Change of Representation and Inductive Bias*. Norwell: Kluwer Academic Publishers.
- Kurshan, R. (1994). *Computer Aided Verification of Coordinating Processes*. Princeton, N.J.: Princeton University Press.
- Laumer, K. (1976). *Bolo, The Annals of the Dinochrome Brigade*. New York, N.Y.: Berkeley Publishing Corp.
- Lichtenstein, O. & Pnueli, A. (1984). Checking that finite state concurrent programs satisfy their linear specifications. *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages* (pp. 271-276).
- Manna, Z. & Pnueli, A. (1991). Completing the temporal picture. *Theoretical Computer Science*, 83(1), 97-130.
- Mitchell, T. (1978). *Version Spaces: An Approach to Concept Learning*. Ph.D. thesis, Stanford University.
- Reps, T. & Teitelbaum, T. (1989). *The Synthesizer Generator*. New York: Springer-Verlag.
- Saberhagen, F. (1967). *Berserkers*. New York: The Berkley Publishing Group.
- Sokolsky, O. & Smolka, S. (1994). Incremental model checking in the modal mu-calculus. *Proceedings of Computer-Aided Verification*.
- Subramanian, D. (1988). *A Theory of Justified Reformulations*. Ph.D. thesis. Stanford University.
- Weld, D., & Etzioni, O. (1994). The First Law of Robotics. *Proceedings of AAAI94* (pp. 1042-1047). AAAI Press.

Gordon, D. (1997a). Asimovian adaptive agents.

Solving a huge number of similar tasks: a combination of multi-task learning and a hierarchical Bayesian approach

Tom Heskes

Foundation for Neural Networks

Geert Grooteplein 21, 6525 EZ Nijmegen, The Netherlands

tom@mbfys.kun.nl

Abstract

In this paper, we propose a machine-learning solution to problems consisting of many similar prediction tasks. Each of the individual tasks has a high risk of overfitting. We combine two types of knowledge transfer between tasks to reduce this risk: multi-task learning and hierarchical Bayesian modeling. Multi-task learning is based on the assumption that there exist features typical to the task at hand. To find these features, we train a huge two-layered neural network. Each task has its own output, but shares the weights from the input to the hidden units with all other tasks. In this way a relatively large set of possible explanatory variables (the network inputs) is reduced to a smaller and easier to handle set of features (the hidden units). Given this set of features and after an appropriate scale transformation, we assume that the tasks are exchangeable. This assumption allows for a hierarchical Bayesian analysis in which the hyperparameters can be estimated from the data. Effectively, these hyperparameters act as regularizers and prevent overfitting. We describe how to make the system robust against nonstationarities in the time series and give directions for further improvement. We illustrate our ideas on a database regarding the prediction of newspaper sales.

- efficient distribution of newspapers and magazines;
- predicting gas consumption of different companies;
- analyzing sales figures of many company branches;
- optimizing stock selection and portfolio management.

The main characteristic of each of these problems is that they are in fact composed of many similar prediction tasks. These individual tasks usually have a low signal-to-noise ratio: in some cases one would be happy if one could explain 10 percent of the variance in the data. Because of the large amount of different tasks, any performance improvement is almost immediately significant, both financially and statistically. Furthermore, in most cases one can easily come up with quite a few (possibly) explanatory variables. For example, in predicting sales figures, one may want to include some of the recent sales figures, sales figures from the same period last year, sales figures from other companies, different kinds of weather information, and so on. Overfitting then becomes a major concern. The question addressed in this paper is therefore: how can we exploit the benefit of not having a single prediction task but a whole set of seemingly similar tasks, such that we can reduce the risk of overfitting in a computationally feasible way?

We propose to combine two approaches: multi-task learning, suggested in the neural-network and machine-learning community, and hierarchical Bayesian modeling, developed in the statistics community. Multi-task learning is treated in Section 2. The idea is that tasks can learn from each other by sharing the same features. The underlying assumption is that such features, typical to the task at hand, indeed exist. Hierarchical Bayesian modeling applies

1 INTRODUCTION

1.1 PROBLEM DESCRIPTION

In this paper, we focus on problems such as

when one can rely on the assumption that *a priori*, i.e., before taking into account the data itself, there is no information to distinguish the model parameters of any one task from those of any of the other tasks. We will describe hierarchical modeling in Section 3.

We will illustrate our ideas on a database concerning the prediction of newspaper sales. This database consists of several years of weekly sales figures for a set of 343 different points of sale. Each point of sale represents a different time-series prediction task. In Section 1.2 we first discuss how to make the tasks “sufficiently similar”, i.e., such that we can apply the approach proposed in Section 2 and 3. Although our examples include collections of time-series tasks, our analysis in these sections is completely static. In Section 4.1 we therefore describe a first crude attempt to handle nonstationarities in the data. Section 4 further links the different components together, recapitulates the assumptions and discusses directions for further improvements.

1.2 MAKING TASKS SIMILAR

The underlying assumption of both the multi-task learning approach and the hierarchical Bayesian approach is that the different tasks can be considered similar. This is not always immediately obvious. As can be seen for example from Figure 1, where we plotted the averages sales of 343 newspaper points of sale versus their standard deviation, the typical number of single copies sold at each outlet ranges from just a few to a few hundred. Still we want to assume that the tasks are, in some sense, exchangeable. In Section 2 this implies that sales figures, when used as explanatory variables, should have more or less the same meaning: 20 newspapers may be quite a lot for a small outlet, but are well below average for a large outlet. Similar reasoning applies to the scaling of model parameters in our choice of prior distributions in Section 3. In the newspaper example, our working hypothesis will be that the points of sale are exchangeable, *after* correcting for their typical scale.

Such a correction can be accomplished by normalizing the sales figures for each outlet separately. The strong correlation between the average sales and the noise level in Figure 1 ($R^2 = 0.90$ on the logarithmic scale) suggests that we can represent the typical scale of each individual outlet through just one parameter θ_i , denoting the average sales of outlet i . We can correct for this typical scale by normalizing all sales figures using this average and the fitted standard deviation as given by the dashed line in Figure 1.

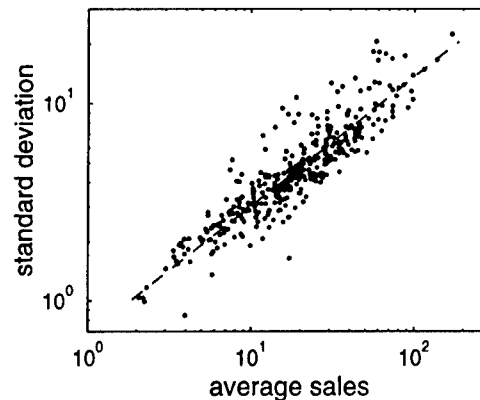


Figure 1: Average newspaper sales θ_i versus the corresponding standard deviation for 343 different points of sale. The dashed line is the least squares fit of the logarithm of the standard deviation as a function of the logarithm of the average sales.

2 MULTI-TASK LEARNING

2.1 ARGUMENTATION

We want to build and train a model relating a set of explanatory variables x to an output z . First we have to choose which explanatory variables to include in such a model. Typically, it is easy to come up with on the order of $n_{\text{inputs}} \approx 20$ input variables (see for example Table 1 where we describe the explanatory variables incorporated in our newspaper example). With on the order of a hundred training patterns per task and a low signal-to-noise ratio, any attempt to fit a direct model between the input variables and the targets corresponding to a single task, is doomed to lead to overfitting and thus lousy prediction performance.

We need some preprocessing stage transforming the n_{inputs} input variables x into a small set of say $n_{\text{features}} \approx 3$ features y , typical to the task at hand. In practice, one often tries to find these features through an iterative process of thinking and testing (see also Figure 4). For example, one tries several ways of combining the most recent sales figures into a single number, tests each of them, and takes the best. Here we propose to *learn* this transformation. We combine all tasks into one big network (see Figure 2). The input units are connected to the hidden (feature) units through a weight matrix B . The weight vector connecting the hidden units to the output unit corresponding to task i is denoted A_i . In other words, all

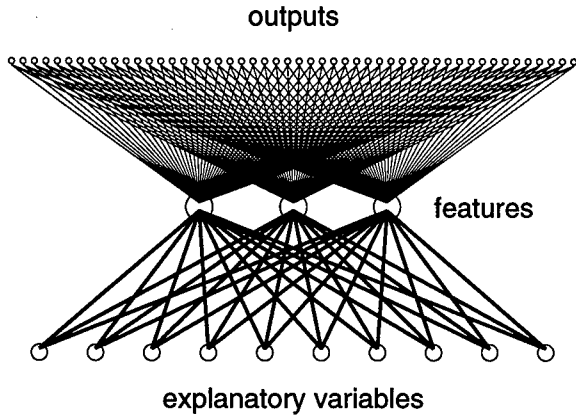


Figure 2: Typical network structure: a reasonably large number of input units, a small number of hidden units, and huge number of output units.

tasks share the weight matrix B , but have independent weight vectors A_i .

In this paper, we will consider the case of linear hidden units. Given an input vector x , the features and outputs are then computed through

$$y_j = \sum_k B_{jk} x_k \text{ and } z_i = A_{i0} + \sum_j A_{ij} y_j, \quad (1)$$

where z_i refers to the output corresponding to task i . We will use A_i to denote the set of all hidden-to-output weights specific to each task, i.e., $A_i \equiv \{A_{i0}, \dots, A_{i, n_{\text{features}}}\}$. We refer to A_i and σ_i as the set of model parameters of task i .

The inputs x can be divided into two categories: those with equal input values across all tasks and those with input values specific to a particular task. Nonspecific inputs in the newspaper example (see Table 1) are e.g. seasonal variables and weather figures (we considered the “average” weather across The Netherlands instead of more local weather figures). The specific inputs should have more or less the same meaning across all tasks. This is accomplished by the transformation of the sales figures described and discussed in Section 1.2. We will use x_i to denote the set of inputs corresponding to task i .

Hidden units do not have bias units: it is easy to see that these can be scaled away into the bias of the output units. We further assume a Gaussian noise model with standard deviation σ_i , which is different for each task, but independent of the inputs x_i . Assuming that

the targets $D_i \equiv \{t_i^\mu\}$ are independently and identically distributed (iid) given the inputs $I_i \equiv \{x_i^\mu\}$, model parameters A_i and σ_i and feature matrix B , we can compute the probability of observing these targets through

$$P(D_i | I_i, A_i, \sigma_i, B) \propto \exp[-E(A_i, \sigma_i, B | D_i, I_i)], \quad (2)$$

where we have defined the error

$$E(A_i, \sigma_i, B | D_i, I_i) = \frac{1}{2} \sum_\mu \left[\frac{(t_i^\mu - z_i^\mu)^2}{\sigma_i^2} - \log \sigma_i \right], \quad (3)$$

with the output z_i^μ computed as in (1). For notational convenience we will from now on leave out the explicit dependency on the inputs I_i . The iid assumption may be too strong for time-series prediction tasks. We will come back to that in Section 4.1.

We propose to find an appropriate feature matrix B through a maximum likelihood procedure: we minimize the error (3), averaged over all n_{tasks} tasks and obtain the maximum likelihood solutions B^{ML} , A_i^{ML} and σ_i^{ML} .

2.2 SIMILAR IDEAS

There has been quite a lot of interesting research in the area of inductive transfer, yielding both empirical and theoretical evidence that multi-task learning improves performance (see [10] for collections of papers on multi-task learning). In [1] the advantage of combining several tasks is investigated theoretically, under the assumption that a feature matrix B common to all tasks indeed exists.

In most approaches to multi-task learning (see e.g. [2] and references therein), all tasks receive the same input information, i.e., all inputs are nonspecific. As in our case, the different tasks are forced to share the same hidden unit representation. Often, but not always, this leads to a better generalization performance [2]. The problems considered in the literature are mostly artificial and combine on the order of 10 or less tasks. An exception is [7], where different tasks concerning stock selection and portfolio management are combined in various ways. This experimental study is probably closest in spirit to our multi-tasking approach, but its number of tasks (36) is still much smaller than the 343 real-world tasks that we use in our simulation.

Group	#	Type	B_1	B_2	B_3
last year sales	3	specific	0.8	54.1	2.9
last year sellouts	3	specific	0.6	1.0	3.6
recent sales	5	specific	93.5	15.4	0.4
recent sellouts	5	specific	1.9	2.9	10.6
weather figures	5	nonspec.	1.2	15.8	15.0
season variables	2	nonspec.	1.9	10.7	67.6

Table 1: List of input variables (see text for further explanation) on the lefthand side. Numbers on the righthand side give the percentage of variance of the features explained by a particular group of input variables.

2.3 FEATURES FOR THE PREDICTION OF NEWSPAPER SALES

The explanatory variables that we took into account are summarized in Table 1. We normalized all non-specific variables. Sales figures were rescaled for each outlet separately as described in Section 1.2. Sellout figures were not rescaled: a sellout is represented by 1, a non-sellout by 0. Recent figures start from 4 weeks ago (the time it takes to collect and administrate all sales figures) and end at 8 weeks ago. Figures from last year are from exactly the same week and the week just before and after that. Weather information includes temperature (relative to the average temperature at the time of year), wind velocity, percentage sunshine, and precipitation (both amount and duration). We slightly changed the definition of the probability model (2) and error (3) to incorporate sellouts (number of sold copies equal to the number of delivered copies) and to take into account that newspaper sales is always integer.

We trained networks with $n_{\text{features}} = 1$ to 8 hidden units. The percentages in Table 1 indicate what part of the variance in each of the features is explained by a particular group of input variables for $n_{\text{features}} = 3$. The features are ordered from most to least relevant. The first feature strongly focuses on the recent sales, the second mostly on the sales from last year, the third mostly on the seasonal variables. Sellouts and weather figures seem to play a minor role, although especially the weather figures explain some of the variance of the second and third feature.

We can also compute the variance in the outputs explained by each group of input variables. The circles in Figure 3 show these percentages for different

numbers of hidden units. With any number of hidden units, the recent sales figures come out to be most relevant. There are, however, interesting differences: a remarkable increase in the relevance of seasonal variables when going from one to two hidden units, a similar increase in the relevance of the recent sellouts when going from two to three hidden units, and somewhat less dramatic increases in last year's figures and weather information.

3 HIERARCHICAL BAYES

3.1 BAYESIAN MODELING

In this section, we replace the maximum likelihood approach of the previous section by a Bayesian approach. We will focus on a Bayesian inference of the model parameters A_{ij} and standard deviations σ_i , given the feature matrix B^{ML} obtained in the previous section. The underlying assumption is that, if there indeed exist features typical to the task of predicting newspaper sales, it should not matter too much whether we find these through an, in this context computationally unfeasible, Bayesian approach or through a much simpler maximum likelihood procedure. Furthermore, we are making lots of other assumptions: our choice of possible explanatory variables, the number of hidden units, the linear transfer function and thus restriction to find linear relationships, and so on. Each set of assumptions corresponds to a different model or hypothesis \mathcal{H} . We can simply include B^{ML} in our definition of \mathcal{H} . In the following, all probability distributions are conditioned on this \mathcal{H} . We will omit this explicit dependency from our notation.

Equation (2) gives the probability distribution of the data for a single task given its model parameters. The probability distribution of all data follows from

$$P(\mathcal{D}|\mathcal{A}) = \prod_i P(D_i|A_i),$$

where A_i now stands for all model parameters of task i (including the standard deviation σ_i), $\mathcal{A} \equiv \{A_1, \dots, A_{n_{\text{tasks}}}\}$, and $\mathcal{D} \equiv \{D_1, \dots, D_{n_{\text{tasks}}}\}$. In a Bayesian analysis, we infer the probability of the model parameters given the data using Bayes' rule:

$$P(\mathcal{A}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{A})P(\mathcal{A})}{P(\mathcal{D})}, \quad (4)$$

where $P(\mathcal{D})$ is a normalization factor independent of the model parameters and $P(\mathcal{A})$ is a prior distribution of the model parameters.

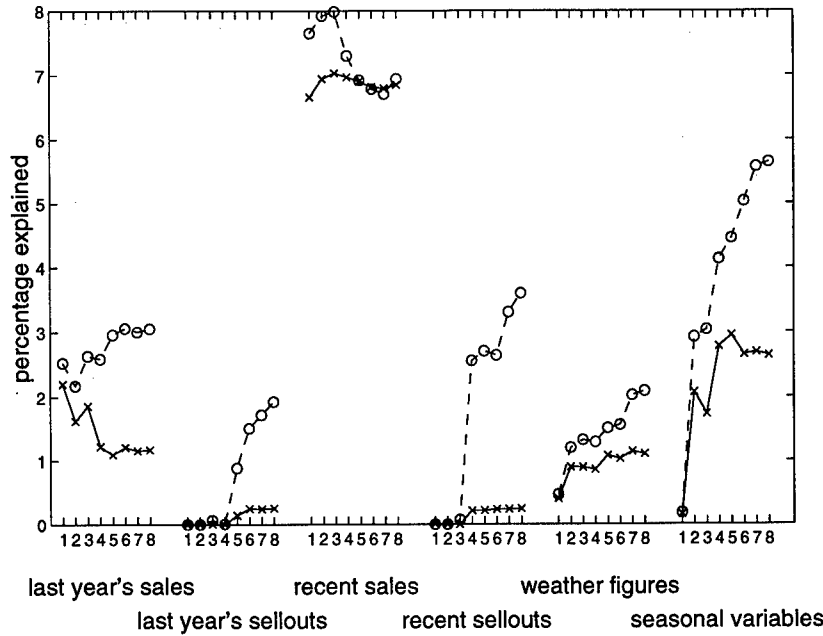


Figure 3: Percentage of variance explained by each group of input variables for various numbers of hidden units: maximum likelihood solution (circles, dashed lines) and most probable solutions (crosses, solid lines).

We take a Gaussian prior on the model parameters $A_i \equiv \{A_{i0}, \dots, A_{i, n_{\text{features}}+1}\}$, with $A_{i, n_{\text{features}}+1} \equiv \log \sigma_i$:

$$P(A_i|\Lambda) \propto \exp \left[-\frac{1}{2} (A_i - m)^T \lambda (A_i - m) \right],$$

where $\Lambda = \{\lambda, m\}$ is called a set of hyperparameters with λ an $[(n_{\text{features}} + 2) \times (n_{\text{features}} + 2)]$ -dimensional symmetric matrix and m an $(n_{\text{features}} + 2)$ -dimensional vector. The model parameters of each task are assumed to be exchangeable, i.e.,

$$P(\mathcal{A}|\Lambda) = \prod_i P(A_i|\Lambda).$$

This exchangeability assumption can be compared with the iid assumption in (2). It implies that, prior to the arrival of data, the probability distribution of the model parameters is invariant under renumbering of the tasks. This is not directly obvious, but may be a reasonable assumption if the outputs for each of the tasks are appropriately rescaled, as discussed in Section 1.2. Another interpretation is that the parameters of the different tasks are penalized by the same set of hyperparameters.

In an exact Bayesian procedure, one should always integrate out the hyperparameters. In a hierarchical

Bayesian procedure, we approximate (4) through

$$P(\mathcal{A}|\mathcal{D}) = \int d\Lambda P(\mathcal{A}|\Lambda, \mathcal{D}) P(\Lambda|\mathcal{D}) \approx P(\mathcal{A}|\Lambda^{\text{MP}}, \mathcal{D}),$$

with $\Lambda^{\text{MP}} = \underset{\Lambda}{\operatorname{argmax}} P(\Lambda|\mathcal{D})$.

The procedure is called hierarchical to indicate that the hyperparameters are inferred at a higher level than the model parameters. The idea behind this approximation is that the distribution $P(\Lambda|\mathcal{D})$ is sharply peaked around its most probable value Λ^{MP} . In our case, where we can use the data for all n_{tasks} tasks to infer the most probable Λ^{MP} , this approximation is extremely accurate and useful. We will simply take an (improper) flat prior for Λ , i.e., $P(\Lambda) \propto 1$, such that the most probable Λ^{MP} is in fact equivalent to the set of maximum likelihood hyperparameters Λ^{ML} .

3.2 RELEVANT LITERATURE

A nice overview of hierarchical (also called empirical) Bayesian modeling, with both a discussion of its underlying assumptions and lots of references to its applications in statistics, can be found in [6]. Our approach is quite similar in spirit to the use of empirical Bayesian techniques in law school validity studies, described and

discussed in [9]. James-Stein estimation can be viewed as the frequentists' equivalent of hierarchical Bayesian modeling. A nice link is provided in [4].

In the neural-network community, hierarchical Bayes is often referred to as the evidence framework [8]. The focus is on learning a single task, where the prior distribution of the weights (usually a diagonal matrix λ and m equal to zero) is chosen to reflect the belief that weights should be small. This yields the Bayesian justification for weight decay or ridge regression. Although from a technical point of view our analysis is at some points quite similar, the meaning of the prior distribution is different: our choice of priors has nothing to do with an *a priori* assumption of small weights, only with exchangeability under a Gaussian probability model.

3.3 INFERENCE OF THE HYPERPARAMETERS

To find the most probable set of hyperparameters Λ^{MP} , we have to maximize the posterior distribution $P(\Lambda|\mathcal{D})$. One way of doing this is through an EM algorithm (see e.g. [6]). The multi-task situation allows for quite a lot of simplifications, which in the end lead to update equations for $\Lambda(n)$. Here we only state the result:

$$\begin{aligned} m(n+1) &= \frac{1}{n_{\text{tasks}}} \sum_i \bar{A}_i(n) \\ \lambda^{-1}(n+1) &= \frac{1}{n_{\text{tasks}}} \sum_i \Sigma_i^2(n) + \\ &\quad \frac{1}{n_{\text{tasks}}} \sum_i [\bar{A}_i(n) - m(n+1)] [\bar{A}_i(n) - m(n+1)]^T \end{aligned}$$

where $\bar{A}_i(n)$ and $\Sigma_i^2(n)$ are the mean and variance of the distribution $P(A|D_i, \Lambda(n))$, respectively. The second term on the righthand side measures the variance between the most probable solutions [given $\Lambda(n)$] for the different tasks, the first term the variance of $P(A|D_i, \Lambda(n))$ around these most probable solutions, averaged over all tasks. We can use Laplace's method (see [6]), based on a quadratic Taylor expansion of $\log P(A|D_i, \Lambda(n))$ around its mode, to find approximations for $\bar{A}_i(n)$ and $\Sigma_i^2(n)$:

$$\begin{aligned} \bar{A}_i(n) &\approx \underset{A}{\operatorname{argmax}} \log P(A|D_i, \Lambda(n)) \\ \text{and } \Sigma_i^2(n) &\approx [H_i(n) + \lambda_n]^{-1}, \end{aligned}$$

where the Hessian matrix $H_i(n)$ of the error $E(A|D_i)$ has to be evaluated at $\bar{A}_i(n)$:

$$H_i(n) \equiv \left. \frac{\partial^2 E(A|D_i)}{\partial A \partial A^T} \right|_{A=\bar{A}_i(n)}.$$

Laplace's method becomes more and more accurate for large sample sizes p per task.

The EM algorithm is intuitive and computationally feasible with the approximation suggested by Laplace's method. A disadvantage of the EM algorithm is that its convergence can be rather slow. A more direct method can be obtained if we make a stronger assumption, namely that the error $E(A|D_i)$ is approximately quadratic in the model parameters A , i.e.,

$$E(A|D_i) \approx E(A_i^{\text{ML}}|D_i) + \frac{1}{2}(A - A_i^{\text{ML}})^T H_i (A - A_i^{\text{ML}}), \quad (5)$$

with A_i^{ML} the maximum likelihood solution minimizing $E(A|D_i)$ and H_i the Hessian evaluated at A_i^{ML} . This is the approximation frequently applied in the evidence framework for neural networks (see e.g. [8]). Now all integrations needed to compute

$$P(\Lambda|\mathcal{D}) \propto \prod_i \int dA P(D_i|A) P(A|\Lambda),$$

are over Gaussian probability distributions, yielding

$$\begin{aligned} \log P(\Lambda|\mathcal{D}) &= -\frac{1}{2} \sum_i (A_i^{\text{ML}} - m)^T Z_i(\lambda) (A_i^{\text{ML}} - m) \\ &\quad + \frac{1}{2} \sum_i \log [\det Z_i(\lambda)], \end{aligned} \quad (6)$$

with $Z_i(\lambda) \equiv (H_i^{-1} + \lambda^{-1})^{-1}$ and where we neglected irrelevant additive constants. The most probable Λ^{MP} maximizes (6) and can be found using e.g. a standard BFGS quasi-Newton algorithm.

3.4 SIMULATIONS

In our newspaper example, the approximation (5) appeared to be extremely accurate. Λ^{MP} was therefore obtained through direct optimization of (6). Given this Λ^{MP} , we computed the most probable model parameters A^{MP} exactly, i.e., without making the approximation (5). The difference between the calculation of the maximum likelihood solutions and the most probable solutions is that the latter are regularized through the hyperparameters Λ^{MP} .

In the previous section we noted dramatic changes in the relevance of groups of input variables with increasing number of hidden units. The relevances for

the most probable solutions, shown by the circles in Figure 3, are surprisingly constant across the different networks with $n_{\text{features}} > 1$: given the correct prior parameters Λ^{MP} , the most probable solutions are roughly the same. Especially the influence of the sell-outs, which seemed to be highly relevant according to the maximum likelihood solutions, almost completely vanishes.

4 DISCUSSION AND CONCLUSION

4.1 DEALING WITH NONSTATIONARITY

Until now, our analysis has been completely static. However, the typical examples given in Section 1 are mostly time-series prediction problems, for which the iid assumption (2) can be too strong. Suppose that we want to predict the output z at “time” μ given inputs x^μ (we leave out the index i for notational convenience). As in the previous sections, we fit the parameter set $\mathbf{A} \equiv \{\theta, A, \sigma\}$ on a training set containing the most recent p patterns. This is a kind of “sliding window approach”: with the addition of every new pattern, the oldest pattern is deleted from the training set. With a delay of n_{delay} patterns between the most recently available pattern and the output to be predicted, the training set ends at $\mu - n_{\text{delay}}$. The naive sliding window approach now computes the output from the input x^μ and the scale and model parameters $\mathbf{A}^{\mu - n_{\text{delay}}}$, which in a way assumes stationarity of the scale and model parameters, i.e., $\mathbf{A}^\mu \approx \mathbf{A}^{\mu - n_{\text{delay}}}$. This naive approach may work fine for many prediction tasks, but leads to lousy predictions on some of them.

To take nonstationarity into account, we add a correction term to the uncorrected prediction:

$$z_{\text{corrected}}^{\mu + n_{\text{delay}}} = z_{\text{uncorrected}}^{\mu + n_{\text{delay}}} + \Delta^\mu.$$

The parameter set \mathbf{A} used to compute the uncorrected $z_{\text{uncorrected}}^\mu$ is determined as before and we still make the assumption that this parameter set is roughly stationary on a time scale of a few patterns. Any nonstationarity should be corrected through Δ^μ . A simple, but efficient procedure for updating Δ^μ is through an exponential smoothing procedure:

$$\bar{\Delta}^\mu = \alpha e_{\text{uncorrected}}^\mu + (1 - \alpha) \bar{\Delta}^{\mu-1} = \alpha e_{\text{corrected}}^\mu + \bar{\Delta}^{\mu-1},$$

with $e_{\text{uncorrected}}^\mu \equiv t^\mu - z_{\text{uncorrected}}^\mu$ and $e_{\text{corrected}}^\mu \equiv t^\mu - z_{\text{corrected}}^\mu$ the difference between the target and

the uncorrected and corrected prediction, respectively. α is a so-called smoothing parameter and $1/\alpha$ corresponds to a typical time scale. It seems reasonable to choose the same α for all tasks. Furthermore, it is well-known (see e.g. [3]) that the precise setting of the smoothing parameter in exponential smoothing hardly affects the prediction performance (see also Figure 4). Perfectly stationary tasks hardly suffer from the extra correction, since their errors $e_{\text{uncorrected}}^\mu$ and $e_{\text{corrected}}^\mu$ tend to average out anyways.

4.2 TEST PERFORMANCE

Some results are displayed in Figure 4. All ideas presented in this paper have been implemented and tested on the prediction of newspaper sales for 343 points of sale. The test set consists of 85 weeks after the training set that has been used for computation of the feature matrix, hyperparameters and most probable model parameters. The model parameters are updated weekly using the sliding windows approach described above. The hyperparameters and feature matrix have been kept constant. The test error is minus the loglikelihood, averaged over both patterns and points of sale.

The network with two hidden units appears to be the best. The regularization through the Bayesian approach cannot completely avoid the risk of overfitting. On the other hand, the best solution without regularization (not shown) is the one with one hidden unit, with a test error of about 2.7, increasing rapidly for more hidden units. The star shows the test performance for a fixed choice of the feature matrix B , made before the start of this project after quite a lot of iterations of thinking, trying and testing. The solution obtained through the multi-tasking approach is significantly better. The righthand side shows the sensitivity to the choice of the smoothing parameter. Taking $\alpha = 0$ is suboptimal: at least for some points of sale, the time series are clearly nonstationary. Any choice of a typical smoothing time between half a year and a year leads to about the same performance.

4.3 STATIONARITY AND SPECIFICITY

A summary of the most important parameters in the complete system is given in Table 2. At the highest level, we have global parameters as the number of features n_{features} and the smoothing parameter α . The choice of these scalar parameters is not extremely critical (see Figure 4) and can be based either on experience with similar databases or by testing a few different alternatives. This is much less the case for the next

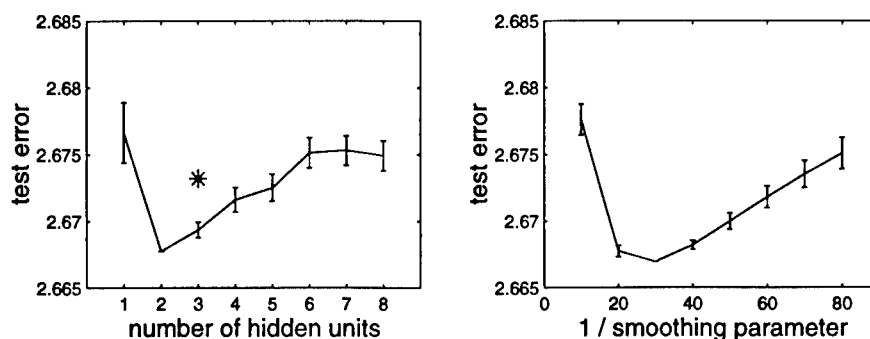


Figure 4: Test error (minus loglikelihood) averaged over 85 weeks and 343 points of sale. Error bars indicate the significance of the difference with the best solution. Left-hand side: as a function of the number of hidden units for smoothing parameter $\alpha = 0.05$. The star corresponds to the performance with a choice of three features obtained after extensive trial and error. Right-hand side: as a function of the typical smoothing time (number of weeks) for the network with two hidden units.

level of parameters: the input-to-hidden weights and the hyperparameters of the prior distribution. These parameters are typical to the task at hand. For example, in predicting newspaper sales, they may be quite different for different days of the week. They can be optimized on a representative set of tasks and kept fixed afterwards. The model and scale parameters as well as the correction terms are obviously specific to each task. We assume that the model parameters are roughly stationary over the length of the training set and can thus be determined through a sliding window approach. The correction terms can be interpreted as corrections to the scale parameters. These may be much less stationary and should be updated with the addition of every single pattern.

4.4 IMPROVEMENTS AND FURTHER DIRECTIONS

Let us recapitulate our approach and underlying assumptions. We started with the observation that we needed some transformation from the possibly quite high-dimensional input space to a much lower dimensional feature space. We proposed to learn this transformation through a maximum likelihood procedure on the weights of a huge network containing all tasks. In this we did not incorporate any prior information, nor did we worry about nonstationarity of the time series involved. Keeping the weights from input to hidden units fixed, we then performed a hierarchical Bayesian analysis to compute hyperparameters, which, roughly speaking, gave us the proper regularization of the model parameters specific to each task. Again,

we disregarded any of the nonstationarity in the data. Finally, keeping both the hyperparameters and input-to-hidden weights fixed, we proposed an exponential smoothing procedure to correct for nonstationarities.

We might try and think of ways to integrate the parts of our approach, instead of applying them sequentially. For example, it may be possible to treat the input to hidden weights as hyperparameters, i.e., at the same level as the hyperparameters Λ for the mean and variance of our prior distribution. The problem here is that it is much more difficult to compute how a change in the hyperparameters of the prior distribution affects the input-to-hidden weights than vice versa. Our treatment follows from the assumption that this effect is negligible for practical purposes. It is not easy to see how to go beyond this simplification, without having to rely on procedures that are computationally unfeasible for any reasonable number of tasks.

About integrating nonstationarity and the Bayesian hierarchical analysis, we may be somewhat more positive. There has been some recent work, which can be viewed as a first attempt to combine Kalman filtering and the Bayesian evidence framework [5]. In this approach the hyperparameters are recomputed every time step. Similar ideas may be applicable to our multi-task situation, although also here we have to worry about the computational feasibility.

Another improvement could be to work with a more complicated prior for the model parameters of the different tasks than the Gaussian considered in this paper. One suggestion is to take another functional form,

Symbol	Description	Time	Tasks	Procedure
n_{features}	number of hidden units	constant	same	experience/test performance
α	smoothing parameter	constant	same	experience/test performance
B	input-to-hidden weights	constant	same	multi-task learning
Λ	hyperparameters	constant	same	Bayesian inference
θ	scale parameters	sliding window	specific	maximum likelihood
A	model parameters	sliding window	specific	MAP estimation
Δ	correction terms	single pattern	specific	exponential smoothing

Table 2: Characteristics of the most important parameters.

for example, a cluster of Gaussians or a prior which forces each task to focus on a subset of the available features. An even more appealing approach would be to make the prior distribution dependent on (known) characteristics of the particular task. In our newspaper case, the width and mean of the distribution could be functions of the distance from the point of sale to the beach, the population density in the vicinity of the point of sale, and so on. The hyperparameters to be inferred from the data would be the parameters in this functional dependency.

Acknowledgement

This research was supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs.

References

- [1] J. Baxter. A Bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, 28:7–39, 1997.
- [2] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.
- [3] C. Chatfield. *The Analysis of Time Series: an Introduction*. Chapman & Hall, London, fourth edition, 1989.
- [4] B. Efron and C. Morris. Data analysis using Stein's estimator and its generalizations. *Journal of the American Statistical Association*, 70:311–319, 1975.
- [5] J. de Freitas, M. Niranjan, and A. Gee. Regularisation in sequential learning algorithms. In *Advances in Neural Information Processing Systems 10*, Cambridge, 1998. MIT Press.
- [6] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian data analysis*. Chapman & Hall, London, 1995.
- [7] J. Ghosn and Y. Bengio. Multi-task learning for stock selection. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 946–952, Cambridge, 1997. MIT Press.
- [8] D. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.
- [9] D. Rubin. Using empirical Bayesian techniques in the law school validity studies (with discussion). *Journal of the American Statistical Association*, 75:801–827, 1980.
- [10] S. Thrun and L. Pratt, editors. *Machine Learning. Second Special Issue on Inductive Transfer*, Dordrecht, 1997. Kluwer Academic.

Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm

Junling Hu and Michael P. Wellman

Artificial Intelligence Laboratory

University of Michigan

Ann Arbor, MI 48109-2110, USA

{junling, wellman}@umich.edu

<http://ai.eecs.umich.edu/people/{junling,wellman}>

Abstract

In this paper, we adopt general-sum stochastic games as a framework for multiagent reinforcement learning. Our work extends previous work by Littman on zero-sum stochastic games to a broader framework. We design a multiagent Q-learning method under this framework, and prove that it converges to a Nash equilibrium under specified conditions. This algorithm is useful for finding the optimal strategy when there exists a unique Nash equilibrium in the game. When there exist multiple Nash equilibria in the game, this algorithm should be combined with other learning techniques to find optimal strategies.

1 Introduction

Reinforcement learning has gained attention and extensive study in recent years [8, 15]. As a learning method that does not need a model of its environment and can be used online, reinforcement learning is well-suited for multiagent systems, where agents know little about other agents, and the environment changes during learning. Applications of reinforcement learning in multiagent systems include soccer [1], pursuit games [17, 4] and coordination games [2]. In most of these systems, single-agent reinforcement learning methods are applied without much modification. Such approach treats other agents in the system as a part of the environment, ignoring the difference between responsive agents and passive environment. In this paper, we propose that a multiagent reinforcement learning method should explicitly take other agents into account. We also propose that a new framework is needed for multiagent reinforcement learning.

The framework we adopt is *stochastic games* (also called *Markov games*) [5, 18], which are the generalization of the Markov decision processes to the case of two or more controllers. Stochastic games are defined as non-cooperative games, where agents pursue their self-interests and choose their actions independently.

Littman [9] has introduced 2-player zero-sum stochastic games for multiagent reinforcement learning. In zero-sum games, one agent's gain is always the other agent's loss, thus agents have strictly opposite interests. In this paper, we adopt the framework of general-sum stochastic games, in which agents need no longer have opposite interests. General-sum games include zero-sum games as special cases. In general-sum games, the notions of "optimality" loses its meaning since each agent's payoff depends on other agents' choices. The solution concept *Nash equilibrium* [11] is adopted. In a Nash equilibrium, each agent's choice is the best response to the other agents' choices. Thus, no agent can gain by unilateral deviation.

we are interested in the Nash equilibrium solution because we want to design learning agent for noncooperative multiagent systems. In such systems, every agent pursues its own goal and there is no communication among agents. A Nash equilibrium is more plausible and self-enforcing than any other solution concept in such systems.

If the payoff structure and state transition probabilities are known to all the agents, we can solve for an Nash equilibrium strategy using a nonlinear programming method proposed by Filar and Vrieze [5]. In this paper, we are interested in situations where agents have incomplete information of other agents' payoff functions and the state transition probabilities. We show that an multiagent Q-learning algorithm can be designed, and it converges to the Nash equilibrium Q values under certain restrictions of the game. Our

algorithm is designed for 2-player general-sum stochastic games, but can be extended to n-player general-sum games.

Our learning algorithm guarantees that an agent can learn a Nash equilibrium. But it does not say whether the other agent will learn the same Nash equilibrium. When there exist only one Nash equilibrium in the game, our learning algorithm works effectively. However, a game can have multiple Nash equilibria. In that case, our learning algorithm needs to be combined with empirical estimation of the action choices of the other agent.

2 Some preliminaries

We state some basic game theory concepts in this section. All concepts here refer to single-state (static) games. In later sections, we will see how the concepts here are connected to multi-state stochastic games.

For zero-sum games, the payoff matrices of two players can be described as $(M, -M)$, since one player's payoff is always the negative of the other. It is sufficient to simplify the game by either M or $-M$. Thus, 2-player zero-sum games are also called *matrix games*. For 2-player general-sum games, the agents' payoff matrices M^1 and M^2 are unrelated. The solutions of the game depend on both M^1 and M^2 . Such games are called *bimatrix games*.

Definition 1 A pair of matrices (M^1, M^2) constitutes a bimatrix game, where M^1 and M^2 are of the same size. The payoff $r^k(a^1, a^2)$ to player k can be found in the corresponding entry of the matrix M^k , $k = 1, 2$. The rows of M^k correspond to actions of player 1, $a^1 \in A^1$. The columns of M^k correspond to actions of player 2, $a^2 \in A^2$. A^1 and A^2 are the sets of discrete actions of players 1 and 2 respectively.

Next, we state some solution concepts for bimatrix games. The main concept is Nash equilibrium [12]. In a Nash equilibrium, each agent's action is the best response to other agents' choices.

Definition 2 A pure strategy Nash equilibrium for bimatrix game G is an action profile (a_*^1, a_*^2) such that

$$\begin{aligned} r^1(a_*, a_*) &\geq r^1(a^1, a_*) \quad \text{for all } a^1 \in A^1 \\ r^2(a_*, a_*) &\geq r^2(a_*, a^2) \quad \text{for all } a^2 \in A^2 \end{aligned}$$

An example of a bimatrix game can be seen in Figure 1, in which the strategy pair (a_1^1, a_1^2) constitutes a pure strategy Nash equilibrium.

$$\begin{array}{cc} & \begin{matrix} M^1 \\ a_1^2 & a_2^2 & a_3^2 \end{matrix} \\ \begin{matrix} a_1^1 \\ a_2^1 \end{matrix} & \begin{pmatrix} 1 & -2 & 4 \\ 0 & 1 & 1 \end{pmatrix} \end{array} \quad \begin{array}{cc} & \begin{matrix} M^2 \\ a_1^2 & a_2^2 & a_3^2 \end{matrix} \\ \begin{matrix} a_1^1 \\ a_2^1 \end{matrix} & \begin{pmatrix} 2 & 1 & 0 \\ 0 & -3 & 2 \end{pmatrix} \end{array}$$

Figure 1: A bimatrix game example

Definition 3 A mixed strategy Nash equilibrium for bimatrix game G is a pair of vectors (ρ_*^1, ρ_*^2) , such that

$$\begin{aligned} \rho_*^1 M^1 \rho_*^2 &\geq \rho^1 M^1 \rho_*^2 \quad \text{for all } \rho^1 \in \sigma(A^1) \\ \rho_*^1 M^2 \rho_*^2 &\geq \rho^1 M^2 \rho_*^2 \quad \text{for all } \rho^2 \in \sigma(A^2) \end{aligned}$$

where $\sigma(A^k)$ is the set of probability distributions over action space A^k , such that for any $\rho^k \in \sigma(A^k)$, $\sum_{a \in A^k} \rho^k(a) = 1$.¹

$\rho^1 M^1 \rho^2 = \sum_{a^1} \sum_{a^2} \rho^1(a^1) r^1(a^1, a^2) \rho^2(a^2)$ is the expected payoff of agent 1 under the situation that player1 and player 2 adopt their mixed strategies ρ^1 and ρ^2 respectively.

The reason we are interested in mixed strategies is that an arbitrary bimatrix game may not have a pure strategy Nash equilibrium, but it always has a mixed strategy Nash equilibrium.

Theorem 1 (Nash, 1951) There exists a mixed strategy Nash equilibrium for any finite bimatrix game.

A mixed strategy Nash equilibrium for any bimatrix game can be found by Mangasarian-Stone algorithm [10], which is a quadratic programming algorithm.

3 Markov Decision Process and reinforcement learning

For comparison purpose, we state the framework of Markov decision process here. Later we can see how the stochastic game framework is related to Markov decision process.

Definition 4 A Markov Decision Process is a tuple $\langle S, A, r, p \rangle$, where S is the discrete state space, A is the discrete action space, $r : S \times A \rightarrow \mathbb{R}$ is the reward function of the agent, and $p : S \times A \rightarrow \Delta$ is the transition function, where Δ is the set of probability distributions over state space S .

¹We abuse the notation a little here. $\rho_*^1 M^1 \rho_*^2$ should be $(\rho_*^1)' M^1 \rho_*^2$, where ρ_*^1 is transposed before being multiplied to the matrix M^1 .

In a Markov decision process, the objective of the agent is to find a strategy (policy) π so as to maximize the expected sum of discounted rewards,

$$v(s, \pi) = \sum_{t=0}^{\infty} \beta^t E(r_t | \pi, s_0 = s) \quad (1)$$

where s_0 is the initial state, r_t is the reward at time t , and $\beta \in [0, 1]$ is the discount factor. We can rewrite Equation (1) as

$$v(s, \pi) = r(s, a_\pi) + \beta \sum_{s'} p(s' | s, a_\pi) v(s', \pi) \quad (2)$$

where a_π is action determined by policy π . It has been proved that there exists an optimal policy π^* such that for any $s \in S$, the following Bellman equation holds:

$$v(s, \pi^*) = \max_a \left\{ r(s, a) + \beta \sum_{s'} p(s' | s, a) v(s', \pi^*) \right\}, \quad (3)$$

where $v(s, \pi^*)$ is called the optimal value for state s .

If the agent knows the reward function and the state transition function, it can solve for π^* by some iterative searching methods [13]. The learning problem arises when the agent does not know the reward function or the state transition probabilities. Now the agent needs to interact with the environment to find out its optimal policy. The agent can learn about the reward function and the state transition function, and then solve for its optimal policy using Equation (3). Such approach is called model-based reinforcement learning. The agent can also directly learn about its optimal policy without knowing the reward function or the state transition function. Such approach is called model-free reinforcement learning. One of the model-free reinforcement learning methods is Q-learning [19].

The basic idea of Q-learning is that we can define the right-hand side of Equation (3) as

$$Q^*(s, a) = r(s, a) + \beta \sum_{s'} p(s' | s, a) v(s', \pi^*) \quad (4)$$

By this definition, $Q^*(s, a)$ is the total discounted reward attained by taking action a in state s and then following the optimal policy thereafter. Then by Equation (3),

$$v(s, \pi^*) = \max_a Q^*(s, a). \quad (5)$$

If we know $Q^*(s, a)$, then the optimal policy π^* can be found, which is always taking an action so as to maximize $Q^*(s, a)$ under any state s .

In Q-learning, the agent starts with arbitrary initial values of $Q(s, a)$ for all $s \in S, a \in A$. At each time t , the agent choose an action and observes its reward r_t . The agent then updates its Q-values based on the following Equation:

$$Q_{t+1}(s, a) = (1 - \alpha_t) Q_t(s, a) + \alpha_t [r_t + \beta \max_b Q_t(s', b)]. \quad (6)$$

where $\alpha_t \in [0, 1]$ is the learning rate. The learning rate α_t needs to decay over time in order for the learning algorithm to converge. Watkins and Dayan [19] proved that sequence (6) converges to the optimal $Q^*(s, a)$.

4 The stochastic game framework

Markov decision process (MDP) is a single agent decision problem. A natural extension of MDP to multiagent systems is stochastic games, which essentially are n-agent Markov decision processes. In this paper, we focus on 2-player stochastic games since they have been well studied.

4.1 Definition of stochastic games

Definition 5 A 2-player stochastic game Γ is a 6-tuple $\langle S, A^1, A^2, r^1, r^2, p \rangle$, where S is the discrete state space, A^k is the discrete action space of player k for $k = 1, 2$, $r^k : S \times A^1 \times A^2 \rightarrow R$ is the payoff function for player k , $p : S \times A^1 \times A^2 \rightarrow \Delta$ is the transition probability map, where Δ is the set of probability distributions over state space S .

To have a closer look at a stochastic game, consider a process that is observable at discrete time points $t = 0, 1, 2, \dots$. At each time point t , the state of the process is denoted by s_t . Assume s_t takes on values from the set S . The process is controlled by 2 decision makers, referred to as player 1 and player 2, respectively. In state s , each player independently chooses actions $a^1 \in A^1, a^2 \in A^2$ and receives rewards $r^1(s, a^1, a^2)$ and $r^2(s, a^1, a^2)$, respectively. When $r^1(s, a^1, a^2) + r^2(s, a^1, a^2) = 0$ for all s, a^1, a^2 , the game is called *zero sum*. When the sum is not restricted to 0 or any constant, the game is called a *general-sum* game.

It is assumed that for every $s, s' \in S$, the transition from s to s' given that the players take actions $a^1 \in A^1$ and $a^2 \in A^2$, is independent of time. That is, there exist stationary transition probabilities $p(s' | s, a^1, a^2)$

for all $t = 0, 1, 2, \dots$, satisfying the constraint

$$\sum_{s'=1}^m p(s'|s, a^1, a^2) = 1, \quad (7)$$

The objective of each player is to maximize a discounted sum of rewards. Let $\beta \in [0, 1)$ be the discount factor, let π^1 and π^2 be the strategies of players 1 and 2 respectively. For a given initial state s , the two players receive the following values from the game:

$$v^1(s, \pi^1, \pi^2) = \sum_{t=0}^{\infty} \beta^t E(r_t^1 | \pi^1, \pi^2, s_0 = s) \quad (8)$$

$$v^2(s, \pi^1, \pi^2) = \sum_{t=0}^{\infty} \beta^t E(r_t^2 | \pi^1, \pi^2, s_0 = s) \quad (9)$$

A strategy $\pi = (\pi_0, \dots, \pi_t, \dots)$ is defined over the whole course of the game. π_t is called the *decision rule* at time t . A strategy π is called a *stationary strategy* if $\pi_t = \bar{\pi}$ for all t , where the decision rule is fixed over time. π is called a *behavior strategy* if $\pi_t = f(h_t)$, where h_t is the history up to time t ,

$$h_t = (s_0, a_0^1, a_0^2, s_1, a_1^1, a_1^2, \dots, a_{t-1}^1, a_{t-1}^2, s_t). \quad (10)$$

A stationary strategy is a special case of behavior strategy when $h_t = \emptyset$.

A decision rule assigns mixed strategies to different states. A decision rule of a stationary strategy has the following form: $\bar{\pi} = (\bar{\pi}(s^1), \dots, \bar{\pi}(s^m))$, where m is the maximal number of states. $\bar{\pi}(s)$ is a mixed strategy under state s .

A Nash equilibrium for stochastic games is defined as following, assuming that the players have complete information about the payoff functions of both players.

Definition 6 In stochastic game Γ , a Nash equilibrium point is a pair of strategies (π_*^1, π_*^2) such that for all $s \in S$

$$v^1(s, \pi_*^1, \pi_*^2) \geq v^1(s, \pi^1, \pi_*^2) \quad \forall \pi^1 \in \Pi^1$$

and

$$v^2(s, \pi_*^1, \pi_*^2) \geq v^2(s, \pi_*^1, \pi^2) \quad \forall \pi^2 \in \Pi^2$$

The definition of Nash equilibrium requires that each agent's strategy is a best response to the other's strategy. Such definition of Nash equilibrium is similar as in other games. The strategies that constitute a Nash equilibrium can be behavior strategies, Markov strategies, or stationary strategies. In this paper, we are

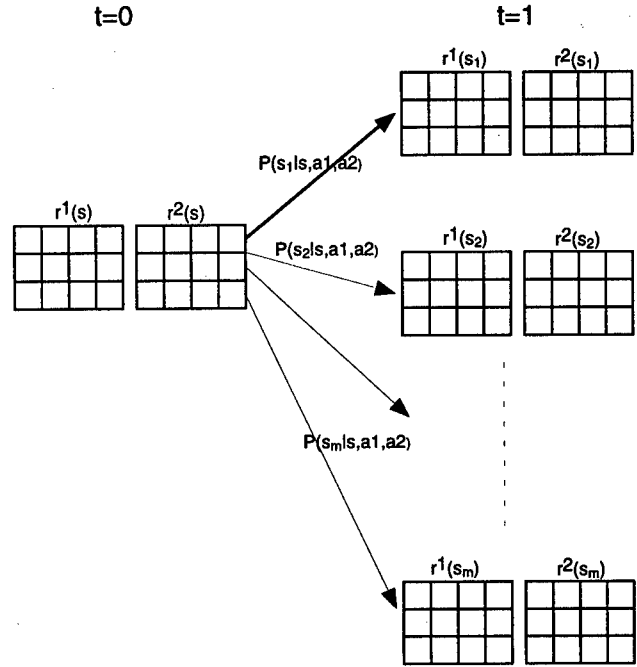


Figure 2: Stochastic games and bimatrix games

interested in stationary strategies, which are the most simple strategies. The following theorem shows that there always exist a Nash equilibrium in stationary strategies for any stochastic game.

Theorem 2 (Filar and Vrieze [5], Theorem 4.6.4) Every general-sum discounted stochastic game possesses at least one equilibrium point in stationary strategies.

4.2 Stochastic games and bimatrix games

We can view each stage of a stochastic game as a bimatrix game, as in Figure 2.

At each time period of a stochastic game, under state s , agent 1 and 2 choose their actions independently and receive their payoffs according to the bimatrix game $(r^1(s), r^2(s))$. Repeated games can be seen as a degenerate case of stochastic games when there is only one state. For example, let \bar{s} be the index of the only state, a repeated game will always have the bimatrix game $(r^1(\bar{s}), r^2(\bar{s}))$ at each time period.

5 Multiagent reinforcement learning

We want to extend traditional reinforcement learning method based on Markov decision process to stochastic games. We assume that our games have incomplete

but perfect information, meaning agents do not know other agents' payoff functions but they can observe other agents' immediate payoffs and actions taken previously.

5.1 Issues in designing a multiagent Q-learning algorithm

The target of our Q-learning is the optimal Q-values, which we define as the following:

$$Q_*^1(s, a^1, a^2) = r^1(s, a^1, a^2) + \beta \sum_{s'=1}^N p(s'|s, a^1, a^2) v^1(s', \pi^1, \pi^2) \quad (11)$$

$$Q_*^2(s, a^1, a^2) = r^2(s, a^1, a^2) + \beta \sum_{s'=1}^N p(s'|s, a^1, a^2) v^2(s', \pi^1, \pi^2) \quad (12)$$

The optimal Q-value of state s and action pair (a^1, a^2) is the total discounted reward received by an agent when both agents execute actions (a^1, a^2) in state s and follow their Nash equilibrium strategies (π^1, π^2) thereafter.

To learn about these Q-values, an agent needs to maintain m Q-tables for its own Q-values, where m is the total number of states. For each agent k , $k = 1, 2$, a Q-table $Q^k(s)$ has its rows corresponding to $a^1 \in A^1$, columns corresponding to $a^2 \in A^2$, and each entry as $Q^k(s, a^1, a^2)$, $k = 1, 2$. The total number of entries agent k needs to learn is $m \times |A^1| \times |A^2|$, where $|A^1|$ and $|A^2|$ are the sizes of action spaces A^1 and A^2 . Assuming $|A^1| = |A^2| = |A|$, then space requirement is $m \times |A|^2$. For n agents, the space requirement is $m \times |A|^n$, which is exponential in the number of agents. Thus for large number of agents, we need to find some compact representation of action space.

As in single-agent Q-learning, the learning agent in multiagent systems updates its Q tables for a given state after it observes the state, actions taken by both agents, and the rewards received by agents. The difference is in the updating rule. In single-agent Q-learning, the Q-values are updated as following,

$$Q_{t+1}(s, a) = (1 - \alpha_t)Q_t(s, a) + \alpha_t[r_t + \beta \max_b Q_t(s', b)].$$

In multiagent Q-learning, we cannot just maximize our own Q-values since the Q-values depend on the action of the other agent.

If it is a zero-sum game, we can minimize over the other agent's actions, and then choose our own maximal after that. This is the minimax-Q learning algorithm in

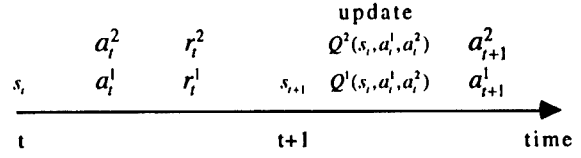


Figure 3: Time line of actions

Littman [9]. For general-sum games, we cannot use mini-max algorithm because the two agent's payoffs are not the opposite of each other. We propose that an agent adopt a Nash strategy to update its Q-values, and this is the best an agent can do in a general-sum game.

5.2 A multiagent Q-learning algorithm

Our Q-learning agent, say agent 1, updates its Q-values according to the following rule:

$$Q_{t+1}^1(s, a^1, a^2) = (1 - \alpha_t)Q_t^1(s, a^1, a^2) + \alpha_t[r_t^1 + \beta \pi^1(s')Q_t^1(s', \pi^2(s'))] \quad (13)$$

where $(\pi^1(s'), \pi^2(s'))$ is a mixed strategy Nash equilibrium for the bimatrix game $(Q_t^1(s'), Q_t^2(s'))$. In order to find out $\pi^2(s')$, agent 1 needs to learn about $Q_t^2(s')$ in the game. The learning is as following:

$$Q_{t+1}^2(s, a^1, a^2) = (1 - \alpha_t)Q_t^2(s, a^1, a^2) + \alpha_t[r_t^2 + \beta \pi^1(s')Q_t^2(s', \pi^2(s'))] \quad (14)$$

Therefore, a learning agent maintains two Q-tables for each state, one for its own Q-values and one for the other agent's. This is possible since we assume an agent can observe the other agent's immediate rewards and previous actions during learning.

The detail of our Q-learning algorithm is stated in Table 1.

When the game is zero-sum, $Q^1(s, a^1, a^2) = -Q^2(s, a^1, a^2) = Q(s, a^1, a^2)$. Thus agent 1 needs to learn only one Q-table for every state. Our Q-learning algorithm becomes,

$$Q_{t+1}(s, a^1, a^2) = (1 - \alpha_t)Q_t(s, a^1, a^2) + \alpha_t[r_t + \beta \max_{\pi^1(s') \in \sigma(A^1)} \min_{\pi^2(s') \in \sigma(A^2)} \pi^1(s')Q_t(s', \pi^2(s'))]$$

This is different from Littman's minimax-Q learning algorithm where Q-value is updated as

$$Q_{t+1}(s, a^1, a^2) = (1 - \alpha_t)Q_t(s, a^1, a^2) + \alpha_t[r_t + \beta \max_{\pi^1(s') \in \sigma(A^1)} \min_{a^2 \in A^2} \pi^1(s')Q_t(s', a^2)]$$

Table 1: Multiagent Q-learning algorithm for Agent 1

Initialize:
 Let $t = 0$,
 For all s in S , a^1 in A^1 , and a^2 in A^2 ,
 let $Q_t^1(s, a^1, a^2) = 1, Q_t^2(s, a^1, a^2) = 1$
 initialize s_0
 Loop
 Choose action a_t^1 based on $\pi^1(s_t)$, which is a mixed strategy Nash equilibrium solution of the bimatrix game $(Q^1(s_t), Q^2(s_t))$.
 Observe r_t^1, r_t^2, a_t^2 , and s_{t+1}
 Update Q^1 , and Q^2 such that
 $Q_{t+1}^1(s, a^1, a^2) = (1 - \alpha_t)Q_t^1(s, a^1, a^2) + \alpha_t[r_t^1 + \beta\pi^1(s_{t+1})Q_t^1(s_{t+1}, \pi^2(s_{t+1}))]$
 $Q_{t+1}^2(s, a^1, a^2) = (1 - \alpha_t)Q_t^2(s, a^1, a^2) + \alpha_t[r_t^2 + \beta\pi^2(s_{t+1})Q_t^2(s_{t+1}, \pi^1(s_{t+1}))]$
 where $(\pi^1(s_{t+1}), \pi^2(s_{t+1}))$ are mixed strategy Nash solutions of the bimatrix game $(Q^1(s_{t+1}), Q^2(s_{t+1}))$
 Let $t := t + 1$

In Littman's Q-learning algorithm, it is assumed that the other agent will always choose a pure Nash equilibrium strategy instead of a mixed strategy.

Another thing to note is that in our Q-learning algorithm, how an agent chooses its action at each time t is not important for the convergence of the learning. But the action choices are important for short-term performance. In this paper, we have not studied the issue of action choice, but will explore it in our future work.

5.3 Convergence of our algorithm

In this section, we prove the convergence of our Q-learning algorithm under certain assumptions. The first two assumptions are standard ones in Q-learning:

Assumption 1 Every state and action have been visited infinitely often.

Assumption 2 the learning rate α_t satisfies the following conditions:

1. $0 \leq \alpha_t < 1, \sum_{t=0}^{\infty} \alpha_t = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$,
2. $\alpha_t(s, a^1, a^2) = 0$ if $(s, a^1, a^2) \neq (s_t, a_t^1, a_t^2)$.

We make further assumptions regarding the structure of the game:

Assumption 3 A Nash equilibrium $(\pi^1(s), \pi^2(s))$ for any bimatrix game $(Q^1(s), Q^2(s))$ satisfies one of the following properties:

1. The Nash equilibrium is global optimal.

$$\pi^1(s)Q^k(s)\pi^2(s) \geq \hat{\pi}^1(s)Q^k(s)\hat{\pi}^2(s) \quad \forall \hat{\pi}^1(s) \in \sigma(A^1), \hat{\pi}^2(s) \in \sigma(A^2), \text{ and } k = 1, 2.$$

2. If the Nash equilibrium is not a global optimal, then an agent receives a higher payoff when the other agent deviates from the Nash equilibrium strategy.

$$\begin{aligned} \pi^1(s)Q^1(s)\pi^2(s) &\leq \pi^1(s)Q^1(s)\hat{\pi}^2(s) \quad \forall \hat{\pi}^2(s) \in \sigma(A^2), \text{ and} \\ \pi^1(s)Q^2(s)\pi^2(s) &\leq \hat{\pi}^1(s)Q^2(s)\pi^2(s) \quad \forall \hat{\pi}^1(s) \in \sigma(A^1). \end{aligned}$$

Our convergence proof is based on the following two Lemmas proved by Szepesvári and Littman [16].

Lemma 1 (Conditional Average Lemma) Under Assumptions 1-2, the process $Q_{t+1} = (1 - \alpha_t)Q_t + \alpha_t w_t$ converges to $E(w_t|h_t, \alpha_t)$, where h_t is the history at time t .

Lemma 2 Under Assumptions 1-2, If the process defined by $U_{t+1}(x) = (1 - \alpha_t(x))U_t(x) + \alpha_t(x)[P_t v^*](x)$ converges to v^* and P_t satisfies $\|P_t V - P_t v^*\| \leq \gamma \|V - v^*\| + \lambda_t$ for all V , where $0 < \gamma < 1$ and $\lambda_t \geq 0$ converges to 0, then the iteration defined by

$$V_{t+1}(x) = (1 - \alpha_t(x))V_t(x) + \alpha_t(x)[P_t V_t](x)$$

converges to v^* .

In order to prove that the convergence point of our Q-learning algorithm is actually the Nash equilibrium point, we need the following theorem proved by Filar and Vrieze [5].

Theorem 3 (Filar and Vrieze [5]) The following assertions are equivalent:

1. For each $s \in S$, the pair $(\pi^1(s), \pi^2(s))$ constitutes an equilibrium point in the static bimatrix game $(Q^1(s), Q^2(s))$ with equilibrium payoffs $(v^1(s, \pi^1, \pi^2), v^2(s, \pi^1, \pi^2))$, and for $k=1, 2$ the entry (a^1, a^2) in $Q^k(s)$ equals

$$Q^k(s, a^1, a^2) = r^k(s, a^1, a^2) + \beta \sum_{s'=1}^N p(s'|s, a^1, a^2) v^k(s', \pi^1, \pi^2).$$

2. (π^1, π^2) is an equilibrium point in the discounted stochastic game Γ with equilibrium payoff $(v^1(\pi^1, \pi^2), v^2(\pi^1, \pi^2))$, where $v^k(\pi^1, \pi^2) = (v^k(s^1, \pi^1, \pi^2), \dots, v^k(s^m, \pi^1, \pi^2))$, $k = 1, 2$.

The above theorem showed that the Nash solution of the bimatrix game $(Q^1(s), Q^2(s))$ defined in Theorem 3 will also be part of the Nash solution for the whole game. If the sequence in our Q-learning algorithm converges to the Q-values defined in Theorem 3, then a pair of stationary Nash equilibrium strategies $(\bar{\pi}^1, \bar{\pi}^2)$ can be derived, where $\bar{\pi}^k = (\bar{\pi}^k(s^1), \dots, \bar{\pi}^k(s^m))$ for $k = 1, 2$. For each state s , $\bar{\pi}^k(s)$ is part of a Nash equilibrium solution of the bimatrix game $(Q^1(s), Q^2(s))$.

Lemma 3 Let $P_t^k Q^k(s) = r_t^k + \beta \pi^1(s) Q^k(s) \pi^2(s)$, $k = 1, 2$, where $(\pi^1(s), \pi^2(s))$ is a pair of mixed Nash equilibrium strategies for the bimatrix game $(Q^1(s), Q^2(s))$. Then $P_t = (P_t^1, P_t^2)$ is a contraction mapping.

Proof. Case 1: $P_t^k Q^k(s) \geq P_t^k \hat{Q}^k(s) \quad \forall k = 1, 2$.

We have

$$\begin{aligned} 0 &\leq P_t^k Q^1(s) - P_t^k \hat{Q}^1(s) \\ &= \beta (\pi^1(s) Q^1(s) \pi^2(s) - \hat{\pi}^1(s) \hat{Q}^1(s) \hat{\pi}^2(s)) \\ &\leq \beta (\pi^1(s) Q^1(s) \pi^2(s) - \pi^1(s) \hat{Q}^1(s) \hat{\pi}^2(s)) \quad (15) \\ &\leq \beta (\pi^1(s) Q^1(s) \hat{\pi}^2(s) - \pi^1(s) \hat{Q}^1(s) \hat{\pi}^2(s)) \quad (16) \\ &= \beta \sum_{a^1} \sum_{a^2} \pi^1(s, a^1) \hat{\pi}^2(s, a^2) (Q^1(s, a^1, a^2) - \hat{Q}^1(s, a^1, a^2)) \quad (17) \\ &\leq \beta \sum_{a^1} \sum_{a^2} \pi^1(s, a^1) \hat{\pi}^2(s, a^2) \|Q^1(s) - \hat{Q}^1(s)\| \\ &= \beta \|Q^1(s) - \hat{Q}^1(s)\|, \end{aligned}$$

where $\|Q^k(s) - \hat{Q}^k(s)\| = \max_{a^1, a^2} |Q^k(s, a^1, a^2) - \hat{Q}^k(s, a^1, a^2)|$. Inequality (15) derives from definition of Nash equilibrium. Inequality (16) is from property 2 of Assumption 2. For cases satisfying property 1 of Assumption 2, the proof is simpler, and we omit it here.

$k = 2$, similar proof as above. Under property 1 of Assumption 2, we have

$$\begin{aligned} 0 &\leq P_t^k Q^2(s) - P_t^k \hat{Q}^2(s) \\ &\leq \beta \sum_{a^1} \sum_{a^2} \pi^1(s, a^1) \pi^2(s, a^2) \|Q^2(s) - \hat{Q}^2(s)\| \\ &= \beta \|Q^2(s) - \hat{Q}^2(s)\|. \end{aligned}$$

Under property 2 of Assumption 2, we have

$$\begin{aligned} 0 &\leq P_t^k Q^2(s) - P_t^k \hat{Q}^2(s) \\ &\leq \beta \sum_{a^1} \sum_{a^2} \hat{\pi}^1(s, a^1) \pi^2(s, a^2) \|Q^2(s) - \hat{Q}^2(s)\| \\ &= \beta \|Q^2(s) - \hat{Q}^2(s)\|. \end{aligned}$$

Case 2: $P_t^k Q^k(s) \leq P_t^k \hat{Q}^k(s)$. Similar proof as in Case 1. For $k = 1$, under property 2 of Assumption 2, we have

$$\begin{aligned} 0 &\leq P_t^k \hat{Q}^1(s) - P_t^k Q^1(s) \\ &\leq \beta \sum_{a^1} \sum_{a^2} \hat{\pi}^1(s, a^1) \pi^2(s, a^2) \|\hat{Q}^1(s) - Q^1(s)\| \\ &= \beta \|\hat{Q}^1(s) - Q^1(s)\|. \end{aligned}$$

Therefore we have $|P_t^k Q^k(s) - P_t^k \hat{Q}^k(s)| \leq \beta \|Q^k(s) - \hat{Q}^k(s)\|$. Since this holds for every state s , we have $\|P_t^k Q^k - P_t^k \hat{Q}^k\| \leq \beta \|Q^k - \hat{Q}^k\|$. \square

Now we proceed to prove our main theorem, which states that the multiagent Q-learning methods converges to the "optimal" (Nash equilibrium) Q values.

Theorem 4 In stochastic game Γ , under Assumptions 1-3, the coupled sequences $\{Q_t^1, Q_t^2\}$, updated by

$$\begin{aligned} Q_{t+1}^k(s, a^1, a^2) &= \\ &(1 - \alpha_t) Q_t^k(s, a^1, a^2) + \alpha_t [r_t^k + \beta \pi^1(s') Q_t^k(s') \pi^2(s')] \quad (18) \end{aligned}$$

where $k = 1, 2$, converge to the Nash equilibrium Q values (Q_*^1, Q_*^2) , with Q_*^k defined as

$$\begin{aligned} Q_*^k(s, a^1, a^2) &= \\ &r^k(s, a^1, a^2) + \beta \sum_{s'=1}^N p(s'|s, a^1, a^2) v^k(s', \pi_*^1, \pi_*^2), \quad (19) \end{aligned}$$

where $(\pi^1(s'), \pi^2(s'))$ is a pair of mixed Nash equilibrium strategies for the bimatrix game $(Q_t^1(s'), Q_t^2(s'))$, function v^k is defined as in (8) and (9), and (π_*^1, π_*^2) is a Nash equilibrium solution for stochastic game Γ .

Proof. By Lemma 3, $\|P_t^k Q^k - P_t^k Q_*^k\| \leq \beta \|Q^k - Q_*^k\|$.

From Lemma 1, the sequence

$$\begin{aligned} Q_{t+1}^k(s, a^1, a^2) &= \\ &(1 - \alpha_t) Q_t^k(s, a^1, a^2) + \alpha_t [r_t^k + \beta \pi^1(s') Q_t^k(s') \pi^2(s')] \end{aligned}$$

converges to

$$\begin{aligned} E(r_t^k + \beta \pi^1 Q^k(s') \pi^2) &= \sum_{s'} P(s'|s, a^1, a^2) \\ &\quad (r^k(s, a^1, a^2) + \beta \pi^1(s') Q^k(s') \pi^2(s')). \end{aligned}$$

Define T^k as

$$(T^k Q^k)(s, a^1, a^2) = \sum_{s'} P(s'|s, a^1, a^2) \left(r^k(s, a^1, a^2) + \beta \pi^1(s') Q^k(s') \pi^2(s') \right)$$

From above, the sequence $\{Q_t^k\}$ converges to $T^k Q^k$. It is easy to show that T^k is a contraction mapping. To see this is true, rewrite T^k as $T^k Q^k(s) = \sum_{s'} P(s'|s, a^1, a^2) P_t Q^k(s)$. Since P_t is a contraction mapping of Q^k and $P(s'|s, a^1, a^2) \geq 0$, T^k is also a contraction mapping of Q^k . We proceed to show that Q_*^k defined in (19) is the fixed point of T^k . From the definition of T^k , we have

$$\begin{aligned} & (T^k Q^k)(s, a^1, a^2) \\ &= \sum_{s'} P(s'|s, a^1, a^2) \left(r^k(s, a^1, a^2) + \beta \pi_*^1(s') Q_*^k(s') \pi_*^2(s') \right) \\ &= r^k(s, a^1, a^2) + \sum_{s'} P(s'|s, a^1, a^2) \beta \pi_*^1(s') Q_*^k(s') \pi_*^2(s') \end{aligned}$$

By Theorem 3, $\pi_*^1(s') Q_*^k(s') \pi_*^2(s') = v^k(s', \pi_*^1, \pi_*^2)$, thus $Q_*^k = T^k Q_*^k$. Therefore the sequence

$$\begin{aligned} Q_{t+1}^k(s, a^1, a^2) &= \\ (1 - \alpha_t) Q_t^k(s, a^1, a^2) + \alpha_t [r_t^1 + \beta \pi_*^1 Q_*^k(s') \pi_*^2] \end{aligned} \quad (20)$$

converges to $T^k Q_*^k = Q_*^k$. By Lemma 2, the sequence (18) converges to Q_*^k . \square

5.4 Discussions

First we want to point out the convergence result does not depend on the sequence of actions taken by either agent. The convergence result only requires that every action has been tried and every state has been visited. It does not require that agent 1 and agent 2 agree on the Nash equilibrium of each bimatrix Q-game during the learning. In fact, agent 1 can learn its optimal Q-value without any behavior assumption of agent 2, as long as agent 1 can observe agent 2's immediate rewards.

Second, the convergence depends on certain restrictions on the bimatrix games during learning. This is required because Nash equilibrium operator is usually not a contraction operator. However, we can probably relax the restriction by proving that a Nash equilibrium operator is a non-expansion operator. Then by the theorem in Szepesvári and Littman [16], the convergence is guaranteed.

6 Future work

There are several issues we have not addressed in this paper. The first is the equilibrium selection problem.

When there exist multiple Nash equilibria, learning one Nash equilibrium strategy does not guarantee the other agent will choose the same Nash equilibrium. Our future work is to combine empirical estimation of the other agent's strategy with reinforcement learning of the Nash equilibrium strategy.

Another issue is related to the action choice during the learning. Even though the multiagent reinforcement learning method converges, it requires infinite trials. During the learning, an agent can choose a myopic action or other kinds of actions. If the agent chooses the action to maximize its current Q-value, its approach is called greedy approach. The drawback of this greedy approach is that the agent may be trapped in a local optimal. To avoid this problem, the agent should explore other possible actions. However, there is cost associated with exploration. By conducting exploration, an agent gives up a better current reward. In our future work, we intend to design an algorithm that can handle exploration and exploitation tradeoff in stochastic games.

References

- [1] Tucker Balch. Learning roles: Behavioral diversity in robot teams. In Sen [14].
- [2] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In Sen [14]. To appear in AAAI-98.
- [3] Eric van Damme. *Stability and Perfection of Nash Equilibria*. Springer-Verlag, 1991.
- [4] Edwin De Jong. Non-random exploration bonuses for online reinforcement learning. In Sen [14].
- [5] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Process*. Springer-Verlag, 1997.
- [6] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. The MIT Press, 1998.
- [7] Junling Hu and Michael P. Wellman. Online learning about other agents in a dynamic multiagent system. To appear in the Proceedings of the Second International Conference on Autonomous Agents, 1998.
- [8] Leslie Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237-285, May 1996.

- [9] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. New Brunswick, 1994.
- [10] O. L. Mangasarian and H. Stone. Two-person nonzero-sum games and quadratic programming. *Journal of Mathematical Analysis and Applications*, 9:348–355, 1964.
- [11] John F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.
- [12] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [13] Martin L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. New York : John Wiley & Sons, 1994.
- [14] Sandip Sen, editor. *Collected papers from the AAAI-97 workshop on multiagent learning*. AAAI Press, 1997.
- [15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press/Bradford Books, March 1998.
- [16] Csaba Szepesvári and Michael L. Littman. A unified analysis of value-function-based reinforcement-learning algorithms. submitted for review, December 1997.
- [17] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, Amherst, MA, June 1993. Morgan Kaufmann.
- [18] Frank Thusijnsman. *Optimality and Equilibria in Stochastic Games*. Amsterdam, the Netherlands : Centrum voor Wiskunde en Informatica, 1992.
- [19] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 3:279–292, 1992.

Coevolutionary Learning: a Case Study

Hugues Juillé

Computer Science Department
Brandeis University
Waltham, Massachusetts 02254-9110
hugues@cs.brandeis.edu

Jordan B. Pollack

Computer Science Department
Brandeis University
Waltham, Massachusetts 02254-9110
pollack@cs.brandeis.edu

Abstract

Coevolutionary learning, which involves the embedding of adaptive learning agents in a fitness environment that dynamically responds to their progress, is a potential solution for many technological chicken and egg problems. However, several impediments have to be overcome in order for coevolutionary learning to achieve continuous progress in the long term. This paper presents some of those problems and proposes a framework to address them. This presentation is illustrated with a case study: the evolution of CA rules. Our application of coevolutionary learning resulted in a very significant improvement for that problem compared to the best known results.

1 Introduction

A recurrent issue in the field of machine learning is that the performance of a learning system relies heavily on the amount of knowledge that has been introduced by the designer. This knowledge can be expressed in the form of an appropriate representation, specific search operators, a training set which provides a good gradient or a special utility function. The success of most learning systems actually results from all this engineering effort.

However, the goal of machine learning is a system that can improve itself by continuously capturing and exploiting new knowledge. The framework which is presented in this paper to achieve such a goal is based on a coevolutionary approach. An important factor in the performance of learning systems is the design of a

training environment. Usually, this training environment is fixed and constructed by the human designer. However, when little knowledge is available about the problem or if this knowledge is difficult to introduce in the training environment, learning can become intractable. The approach proposed in this paper to get round that problem consists of coevolving the training environment with a population of learners. Starting with simple problems, the training environment gets more challenging as learners are improving themselves. Hopefully, such a setup leads to continuous progress. For the rest of the paper, we define *coevolutionary learning* as a search procedure involving a population of learners coevolving with a population of problems such that *continuous progress* results from this interaction.

In practice, the picture is not that simple. We will discuss the different issues that are involved to achieve coevolutionary learning by considering a particular problem: the discovery of cellular automata rules to implement a classification task. This problem presents some interesting properties that provide us with a simple framework to monitor the dynamics of the search resulting from different setups. Section 2 describes this problem. In section 3, an experimental analysis presents the different impediments to coevolutionary learning and a solution to address them is proposed in section 4. Experimental results for the classification problem are presented in section 5.

2 Description of the Problem

2.1 One-Dimensional Cellular Automata

A one-dimensional cellular automaton (CA) is a linear wrap-around array composed of N cells in which each cell can take one out of k possible states. A rule is

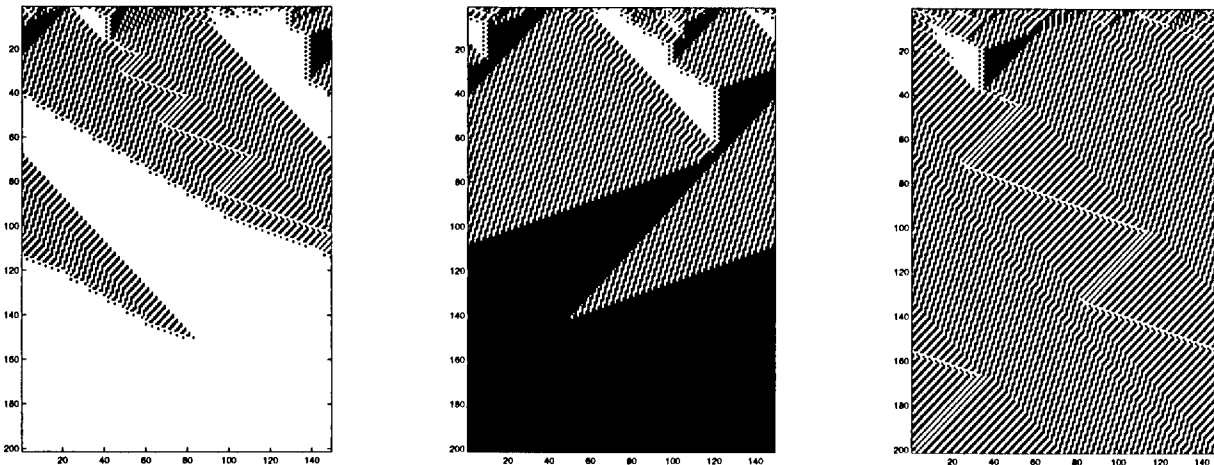


Figure 1: Three space-time diagrams describing the evolution of CA states: in the first two, the CA relaxes to the correct uniform pattern while in the third one it doesn't converge at all to a fixed point.

Table 1: Performance of different published CA rules and a new best rule for the $\rho_c = 1/2$ task.

N	149	599	999
Coevolution	0.863 +/- 0.001	0.822 +/- 0.001	0.804 +/- 0.001
Das rule	0.823 +/- 0.001	0.778 +/- 0.001	0.764 +/- 0.001
ABK rule	0.824 +/- 0.001	0.764 +/- 0.001	0.730 +/- 0.001
GKL rule	0.815 +/- 0.001	0.773 +/- 0.001	0.759 +/- 0.001

defined for each cell in order to update its state. This rule determines the next state of a cell given its current state and the state of cells in a predefined neighborhood. For the model discussed in this paper, this neighborhood is composed of cells whose distance is at most r from the central cell. This operation is performed synchronously for all the cells in the CA. From now on, we will consider that the state of cells is binary ($k = 2$), $N = 149$ and $r = 3$. This means that the size of the rule space is $2^{2^{2 \cdot r + 1}} = 2^{128}$.

Cellular automata have been studied widely as they represent one of the simplest systems in which complex emergent behaviors can be observed. This model is very attractive as a means to study complex systems in nature. Indeed, the evolution of such systems is ruled by simple, locally-interacting components which result in the emergence of global, coordinated activity.

2.2 The Majority Function

This is a density classification task, for which one wants the state of the cells of the CA to relax to all 0's or all 1's depending on the density of the initial configuration (IC) (whether it has more 0's or more 1's), within a maximum of M time steps. Following

[Mitchell et al., 1994], ρ_c denotes the threshold for the classification task (here, $\rho_c = 1/2$), ρ denotes the density of 1's in a configuration and ρ_0 denotes the density of 1's in the initial configuration. Figure 1 presents three examples of the space-time evolution of a CA. One with $\rho_0 < \rho_c$ on the left and another with $\rho_0 > \rho_c$ in the middle for which the CA relaxes to the correct configuration. The third one shows an instance for which the CA doesn't relax to any of the two desired convergence patterns. For each diagram, the initial configuration is at the top and the evolution in time of the state of the CA is represented downward.

The task $\rho_c = 1/2$ is known to be difficult. In particular, it has been proven that no rule exists that results in the CA relaxing to the correct state for all possible ICs [Land & Belew, 1995]. Indeed, the density is a global property of the initial configuration while individual cells of the CA have access to local information only. Discovering a rule that will display the appropriate computation by the CA with the highest accuracy is a challenge, and the upper limit for this accuracy is still unknown. Table 1 describes the performance for that task for different published rules and different values of N , along with the performance of the new best rule that resulted from the work

presented in this paper. The Gacs-Kurdyumov-Levin (GKL) rule was designed in 1978 for a different goal than solving the $\rho_c = 1/2$ task [Mitchell et al., 1994]. However, for a while it provided the best known performance. [Mitchell et al., 1994] and [Das et al., 1994] used Genetic Algorithms (GAs) to explore the space of rules. The main purpose of this work was to develop a particle-based methodology for the analysis of the complex behaviors exhibited by CAs. The GKL and Das rules are human-written while the Andre-Bennett-Koza (ABK) rule has been discovered using the Genetic Programming paradigm [Andre et al., 1996]. More recently, [Paredis, 1997] describes a coevolutionary approach to search the space of rules and shows the difficulty of coevolving consistently two populations towards continuous improvement. [Capcarrere et al., 1996] also reports that by changing the specification of the convergence pattern, a two-state, $r = 1$ CA exists that can perfectly solve the density problem in $\lceil N/2 \rceil$ time steps.

For the $\rho_c = 1/2$ task, it is believed that the best rules are in the domain of the rule space with density close to 0.5. An intuitive argument to support this hypothesis is presented in [Mitchell et al., 1993]. It is also believed that the most difficult ICs are those with density close to 0.5.

3 Models for Coevolutionary Search

The idea of using coevolution in search was introduced by [Hillis, 1992]. In coevolution, individuals are evaluated with respect to other individuals instead of a fixed environment (or landscape). As a result, agents adapt in response to other agents' behavior. The particular model of coevolution considered in this paper is based on two populations for which the fitness of individuals in each population is defined with respect to the members of the other population. Two cases can be considered in such a framework, depending on whether the two populations benefit from each other or whether they have different interests. Those two modes of interaction are called *cooperative* and *competitive* respectively. In the following sections, those modes of interaction are described experimentally using the $\rho_c = 1/2$ task in order to stress the different issues related to coevolutionary learning.

For the experiments presented in this section, we used an implementation of Genetic Algorithms similar to the one described in [Mitchell et al., 1994]. Each rule is coded on a binary string of length $2^{2*r+1} = 128$. One-point crossover is used with a 2% bit mutation

probability. The population size is $n_R = 200$ for rules and $n_{IC} = 200$ for ICs. The population of ICs is composed of binary strings of length $N = 149$. The population of rules and ICs are initialized according to a uniform distribution over $[0.0, 1.0]$ for the density. For all the experiments in this paper, the value of M (the maximum number of time steps) is set to 320 and is kept unchanged. At each generation, the top 95% of each population reproduces to the next generation and the remaining 5% is the result of crossover between parents from the top 95% selected using a fitness proportionate rule. This small generation gap (the percentage of new individuals) has been used because of the dynamic fitness landscape. Indeed, a large generation gap can result in a dramatic change in the composition of the population. As a consequence, because of the relative definition of the fitness, a lot of variation in individuals' fitness can occur from one generation to the other, making the identification of the most promising individuals very unreliable.

3.1 Cooperation between Populations

In this mode of interaction, improvement on one side results in positive feedback on the other side. As a result, there is a reinforcement of the relationship between the two populations. From a search point of view, this can be seen as an *exploitative* strategy. Agents are not encouraged to explore new areas of the search space but only to perform local search in order to further improve the strength of the relationship. In the cooperative model, a natural definition for the fitness of rules (resp. ICs) is the number of ICs (resp. rules) for which the CA relaxes to the correct state:

$$f(R_i) = \sum_{j=1}^{n_{IC}} \text{covered}(R_i, IC_j)$$

$$f(IC_j) = \sum_{i=1}^{n_R} \text{covered}(R_i, IC_j)$$

where $\text{covered}(R_i, IC_j)$ returns 1 if a CA using rule R_i and starting from initial configuration IC_j relaxes to the correct state. Otherwise, it returns 0.

Figure 2 presents the evolution of the density of rules and ICs for one run using this cooperative model. Without any surprise, the population of rules and ICs quickly converge to a domain of the search space where ICs are easy for rules and rules consistently solve ICs. As a result, there is little exploration of the search space. The convergence configuration depends on the initial populations, some other runs ended up with low

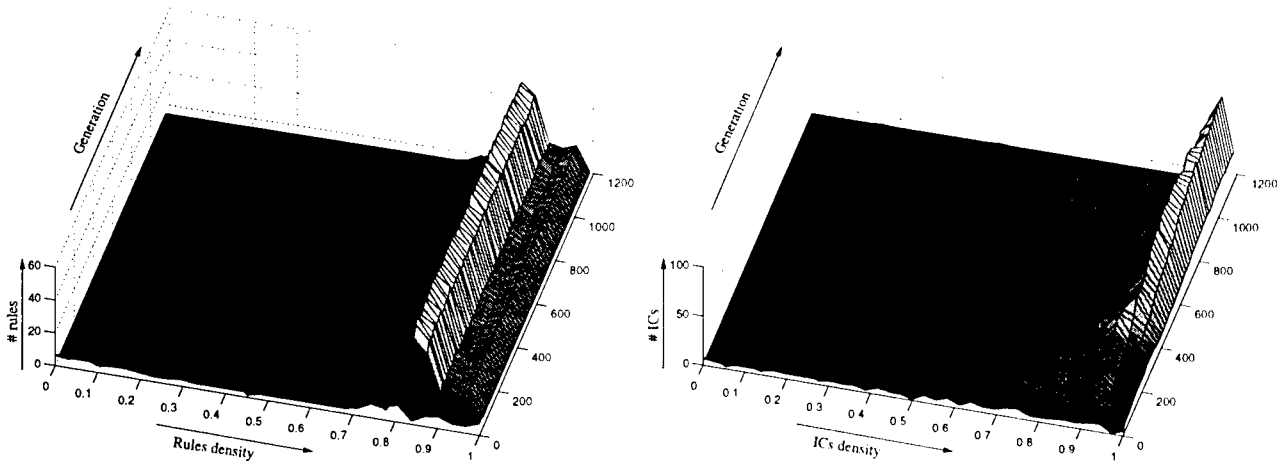


Figure 2: Coevolution of CA rules (left) and ICs (right) in a cooperative relationship.

density rules and ICs. This experiment confirms that ICs with low or high density are the easiest to classify since a larger number of rules classify them correctly.

3.2 Competition between Populations

In this mode of interaction, the two populations are in conflict. Improvement on one side results in negative feedback for the other population. The fitness of rules and ICs defined in the cooperative case can be modified as follows to implement the competitive model:

$$f(R_i) = \sum_{j=1}^{n_{IC}} \text{covered}(R_i, IC_j)$$

$$f(IC_j) = \sum_{i=1}^{n_R} \frac{1}{\text{covered}(R_i, IC_j)}$$

where $\text{covered}(R_i, IC_j)$ returns the inverse of the original function. Here, the goal of rules is to defeat (i.e. cover) ICs, while the goal of ICs is to defeat rules by discovering initial configurations that are difficult to classify. Figure 3 describes a run using this definition of the fitness. Two kind of behaviors can be observed in this experiment. In a first stage, the two populations exhibit a cyclic behavior. It is a consequence of the *Red Queen* effect [Cliff & Miller, 1995]: fitness landscapes are changing as a result of agents of each population adapting in response to the evolution of members of the other population. The evaluation of individuals' performance in this changing environment makes continuous progress difficult. A typical consequence is that agents have to learn again what they already knew in the past. In the context of evolutionary search, this means that domains of the state

space that have already been explored in the past are searched again. Then, a stable state is reached: in this case, the population of rules adapts faster than the population of ICs, resulting in a population focusing only on rules with high density and eliminating all instances of low density rules (a finite population is considered). Then, low density ICs exploit those rules and overcome the entire population. A similar experiment is described in [Paredis, 1997].

3.3 Resource Sharing and Mediocre Stable States

Several techniques have been designed to improve evolutionary search. Usually they maintain diversity in the population in order to avoid premature convergence. [Mahfoud, 1995] presents different niching techniques that achieve this goal. Resource sharing, first introduced in [Rosin & Belew, 1995], is a technique that we successfully used in the past [Juillé & Pollack, 1996]. Resource sharing implements a coverage-based heuristic by giving a higher payoff to problems that few individuals can solve. Resource sharing can be introduced in the competitive model of coevolution as follows:

$$f(R_i) = \sum_{j=1}^{n_{IC}} \text{weight_IC}_j \times \text{covered}(R_i, IC_j)$$

where:

$$\text{weight_IC}_j = \frac{1}{\sum_{k=1}^{n_R} \text{covered}(R_k, IC_j)}$$

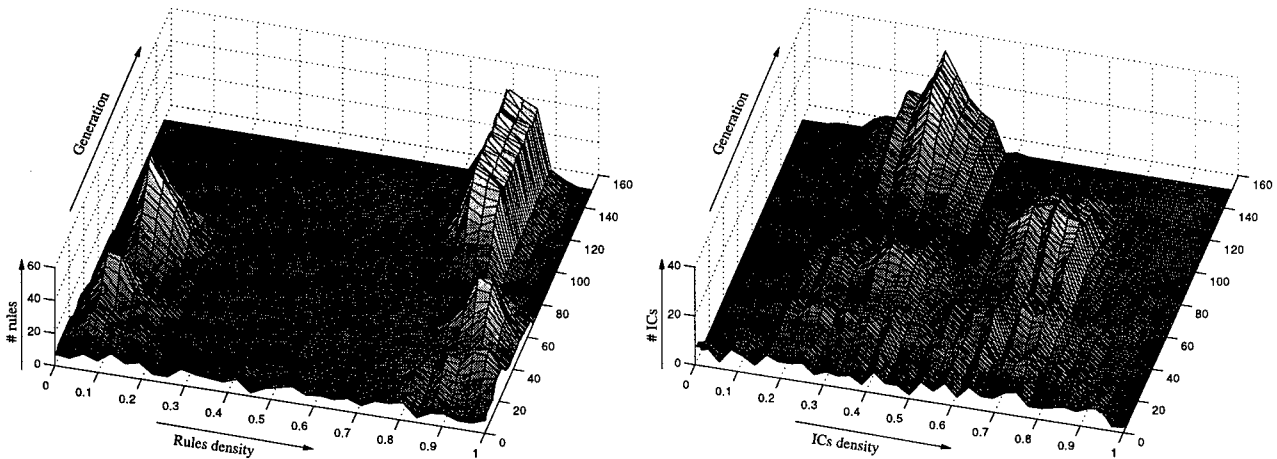


Figure 3: Coevolution of CA rules (left) and ICs (right) in a competitive relationship.

and

$$f(IC_j) = \sum_{i=1}^{n_R} \text{weight_}R_i \times \overline{\text{covered}(R_i, IC_j)}$$

where:

$$\text{weight_}R_i = \frac{1}{\sum_{k=1}^{n_{IC}} \overline{\text{covered}(R_i, IC_k)}}$$

In this definition, the weight of an IC corresponds to the payoff it returns if a rule covers it. If few rules cover an IC, this weight will be much larger than if a lot of rules cover that same IC. The definition for the weight of rules has the same purpose. This framework allows the presence of multiple niches (or species) in populations. Figure 4 describes one run for this definition of the fitness. The cyclic behavior which was observed in the previous section doesn't occur anymore. Instead, two species coexist in the population of rules: a species for low density rules and another one for high density rules. Those two species drive the evolution of ICs towards the domain of initial configurations that are most difficult to classify (i.e., $\rho_0 = 1/2$). However, the two populations have entered a *mediocre stable state*. This means that multiple average performance niches coexist in both populations in a stable manner. Put in another way, this can be seen as an equilibrium configuration in which a number of suboptimal species have found a way to collude by sharing the total credit between themselves. Usually, this is a consequence of some singularities inherent in the problem definition and/or the search procedure. In our example, ICs are concentrated around the $\rho_0 = 1/2$ threshold and they can be divided into two groups: those with density $\rho_0 < 1/2$ and those with density $\rho_0 > 1/2$. This distribution means that ICs can be exploited consistently

by rules with low and high density that both occur in the second population (because a CA implementing a low (resp. high) density rule usually relaxes to all 0's (resp. all 1's) for most ICs). However, this is a mediocre stable state in the sense that evolved rules have poor performance with respect to the $\rho_c = 1/2$ task and there is no pressure towards improvement. The concept of mediocre stable states is also discussed in [Pollack et al., 1996].

3.4 Discussion

We have described different models for the coevolution of two populations. Some of the fundamental impediments to coevolutionary learning have been identified along with some of the reasons why continuous progress is difficult to achieve. It is now clear that none of these approaches can address successfully the problem of coevolutionary learning alone. All the rules discovered in those experiments perform poorly since they never approach the 50% density. The following section proposes a framework to get around those problems.

Each of the canonical models discussed so far implements a single specific strategy. In the literature, there has been some successful applications for both the cooperative and the competitive approaches. However, those works usually introduce some mechanisms to address the problems specific to each model. For instance, a noisy evaluation of the fitness can force exploration in a cooperative model, and an evaluation of individuals with respect to a set of opponents extracted from previous generations can limit the cyclic behavior observed in competitive models (e.g., see the life-time fitness evaluation technique described in

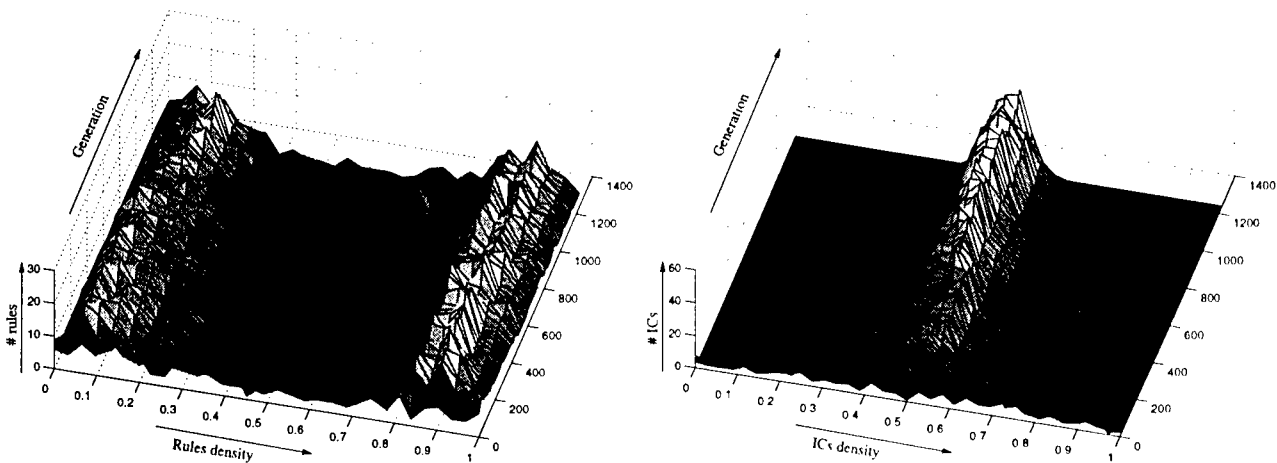


Figure 4: Coevolution of CA rules (left) and ICs (right) in a competitive relationship with resource sharing.

[Paredis, 1996] or the “hall of fame” method presented in [Rosin, 1997]). However, those mechanisms usually fail to address entirely the fundamental issues discussed previously.

4 Coevolving the “Ideal” Trainer

4.1 Presentation of our Approach

From the analysis of the experiments presented in section 3 at least two reasons seem to prevent continuous progress in coevolutionary search. The first one is that the training environment provided by the population of ICs returns little information to the population of evolving rules because a stable configuration is reached in which the credit is distributed according to a fixed pattern (e.g., all the ICs are covered by rules). The second reason is that the dynamics of the search performed by the two coevolving populations doesn't drive individuals to the domain of the state space that contains most promising solutions because there is no “high-level” strategy to play that role. This is a consequence of the Red Queen effect.

Our approach proposes a coevolutionary framework in which those two issues are addressed as follows:

- the training environment provides at any time a gradient for search by proposing a variety of problems covering a range of difficulty. Indeed, if problems are too difficult, nobody can solve them. On the contrary, if they are too easy, everybody can solve them. In both cases, those problems are useless for learning since they provide little feedback.
- a “high-level” strategy allows continuous progress

by preventing the negative effects associated with the Red Queen.

The central idea of this coevolutionary learning approach consists in exposing learners to problems that are just beyond those they know how to solve. By maintaining this constant pressure towards slightly more difficult problems, an arms race among learners is induced such that learners that adapt better have an evolutionary advantage. The underlying heuristic implemented by this arms race is that *adaptability* is the driving force for improvement. The difficulty resides in the accurate implementation of the concepts presented above in a search algorithm. So far, our methodology to implement such a system consists in the construction of an explicit topology over the space of problems by defining a partial order with respect to the relative difficulty of problems among each other. In our current work, the concept of “relative difficulty” has been defined by exploiting some *a priori* knowledge about the task. The definition of this topology over the space of problems makes possible the implementation of the two goals required in our coevolutionary learning approach. Indeed, since learners are evaluated against a known range of difficulty for problems, it is possible to monitor their progress and to expose them to problems that are just “a little more difficult”. In our work, this last concept has been formalized by defining empirically a distance measure. In this framework, learners are always exposed to a gradient for search and it is possible to control the evolution of the training environment towards more difficult problems in order to ensure continuous progress.

In the future, our goal is to eliminate some of those ex-

explicit components by introducing some heuristics that automatically identify problems that are appropriate for the current set of learners. The work of Rosin [Rosin, 1997] already describes some methods to address this issue.

4.2 Discussion

As stated previously, the coevolutionary learning framework introduces a pressure towards adaptability. The central assumption is that individuals that adapt faster than others in order to solve the new challenges they are exposed to are also more likely to solve even more difficult problems. The main difficulty is to setup a coevolutionary framework that implements this heuristic accurately and efficiently.

The new contribution of this work is the idea of maintaining a gradient for search as one of the underlying heuristics. In the literature, different approaches have been proposed to address the issues associated with the Red Queen effect [Paredis, 1996, Rosin, 1997]. However, to our knowledge, explicit methods to force progress and to prevent mediocre stable states in the context of evolutionary search have never been tried.

The idea of introducing a pressure towards adaptability as the central heuristic for search is not new. Schmidhuber [Schmidhuber, 1995] proposed the Incremental Self-Improvement system in which adaptability is the measure that is optimized. The concept of an ideal trainer is also discussed in [Epstein, 1994] in the context of game learning. However, this work addresses the issue of designing the "ideal" training procedure which would result in high quality players rather than coevolving the training environment in response to the progress of learners.

5 Application to the Discovery of CA Rules

5.1 Experimental Setup

The approach described in the previous section is applied to the $\rho_c = 1/2$ task. It is believed that ICs become more and more difficult to classify correctly as their density gets closer to the ρ_c threshold. Therefore, our idea is to construct a framework that adapts the distribution of the density for the population of ICs as CA-rules are getting better to solve the task. The following definition for the fitness of rules and ICs has

been used to achieve this goal.

$$f(R_i) = \sum_{j=1}^{n_{IC}} \text{weight_}IC_j \times \text{covered}(R_i, IC_j)$$

where:

$$\text{weight_}IC_j = \frac{1}{\sum_{k=1}^{n_R} \text{covered}(R_k, IC_j)}$$

and

$$f(IC_j) = \frac{\sum_{i=1}^{n_R} \text{weight_}R'_i \times E(R_i, \rho(IC_j))}{\text{covered}(R_i, IC_j)}$$

where:

$$\text{weight_}R'_i = \frac{1}{\sum_{k=1}^{n_{IC}} E(R_i, \rho(IC_k)) \times \text{covered}(R_i, IC_k)}$$

This definition implements the competitive relationship with resource sharing. However, a new component, namely $E(R_i, \rho(IC_j))$, has been added in the definition of the ICs' fitness. The purpose of this new component is to penalize ICs with density $\rho(IC_j)$ if little information is collected with respect to the rule R_i . Indeed, we consider that if a rule R_i has a 50% classification accuracy over ICs with density $\rho(IC_j)$ then this is equivalent to random guessing and no payoff should be returned to IC_j . On the contrary, if the performance of R_i is significantly better or worse than the 50% threshold for a given density of ICs this means that R_i captured some relevant properties to deal with those ICs. Once again, the idea is that the training environment should be composed of ICs that provide useful information to identify good rules from poor ones. In order to allow continuous progress, our implementation exploits an intrinsic property of the $\rho_c = 1/2$ task. Indeed, it seems that CA-rules that cover ICs with density $\rho_0 < 1/2$ (resp. $\rho_0 > 1/2$) with high performance will also be very successful over ICs with density $\rho'_0 < \rho_0$ (resp. $\rho'_0 > \rho_0$). Therefore, as ICs become more difficult, their density is approaching $\rho_0 = 1/2$ but rules don't have to be tested against easier ICs. Following this idea, we defined $E()$ as the complement of the entropy of the outcome between a rule and ICs with a given density:

$$E(R_i, \rho(IC_j)) = \log(2) + p \log(p) + q \log(q)$$

where: p is the probability that an IC with density $\rho(IC_j)$ defeats the rule R_i and $q = 1 - p$. $E()$ implements the distance measure discussed in section 4.1. Its purpose is to maintain the balance between the search for more difficult ICs and ICs that can be solved by rules. In practice, the entropy is evaluated by performing some statistics over the population of ICs.

Table 2: Description of the current best rule and published rules for the $\rho_c = 1/2$ task.

Coevolution	00010100	01011111	01000000	00000000	00010111	11111100	00000010	00010111
	00010100	01011111	00000011	00001111	00010111	11111111	11111111	11010111
Das rule	00000111	00000000	00000111	11111111	00001111	00000000	00001111	11111111
	00001111	00000000	00000111	11111111	00001111	00110001	00001111	11111111
ABK rule	00000101	00000000	01010101	00000101	00000101	00000000	01010101	00000101
	01010101	11111111	01010101	11111111	01010101	11111111	01010101	11111111
GKL rule	00000000	01011111	00000000	01011111	00000000	01011111	00000000	01011111
	00000000	01011111	11111111	01011111	00000000	01011111	11111111	01011111

5.2 Experimental Results

Experiments were performed with different sizes for the population of rules and ICs. The best rule whose performance is reported in table 1 resulted from the experiments that used the largest population size. In those experiments, 6 runs were performed for 5,000 generations, using a size of 1,000 for the two populations. Each rule is coded on a binary string of length $2^{2*r+1} = 128$. One-point crossover is used with a 2% bit mutation probability. The population of rules is initialized according to a uniform distribution over $[0.0, 1.0]$ for the density. Each individual in the population of ICs represents a density $\rho_0 \in [0.0, 1.0]$. This population is also initialized according to a uniform distribution over $\rho_0 \in [0.0, 1.0]$. At each generation, each member generates a new instance for an initial configuration with respect to the density it represents. All rules are evaluated against this new set of ICs. The generation gap is 5% for the population of ICs (i.e., the top 95% ICs reproduce to the next generation). There is no crossover nor mutation. The new 5% ICs are the result of a random sampling over $\rho_0 \in [0.0, 1.0]$ according to a uniform probability distribution. The generation gap is 80% for the population of rules. New rules are created by crossover and mutation. Parents are randomly selected from the top 20%. All runs consistently evolved some rules that score above 82%. Table 2 describes lookup tables for the current best CA rule and other rules discussed in the literature. The leftmost bit corresponds to the result of the rule on input 00000000, the second bit corresponds to input 00000001, ... and the rightmost bit corresponds to input 11111111.

Figure 5 describes the evolution of the density of rules and ICs for one run. As rules improve, their density gets closer to $1/2$ and the density of ICs is distributed on two peaks on each side of $\rho_c = 1/2$. In that particular run, it is only after 1,300 generations that a significant improvement is observed for rules and that, in response, the population of ICs adapts dramatically

in order to propose more challenging initial configurations. This shows that our strategy to coevolve the training environment and the learners has been successfully implemented in the definition of the fitness functions.

6 Conclusion

This paper presents a new framework based on the concept of *cocvolutionary learning*. This approach coevolves the training environment with respect to a population of learners such that learners are always exposed to a gradient for search, and evolution of problems towards increasing difficulty is maintained. The work presented in this paper addresses those issues by defining a topology over the space of problems. Then, a procedure is implemented such that the training environment automatically adapts in response to the progress of learners by proposing more challenging problems. We applied this framework to the problem of evolving CA rules for a classification task. Our experiments resulted in a new rule whose performance improves very significantly over previously known rules for that particular task.

Acknowledgment

I would like to thank Melanie Mitchell for her help and useful discussions.

References

- [Andre et al., 1996] Andre, D., Bennett III, F. H., & Koza, J. R. (1996). Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Proceedings of the Fifth Artificial Life Conference*, pp. 16–18.
- [Capcarrere et al., 1996] Capcarrere, M. S., Sipper, M., & Tomassini, M. (1996). Two-state, $r=1$ cellular automaton that classifies density. *Physical Review Letters*, 77(24):4969–4971.

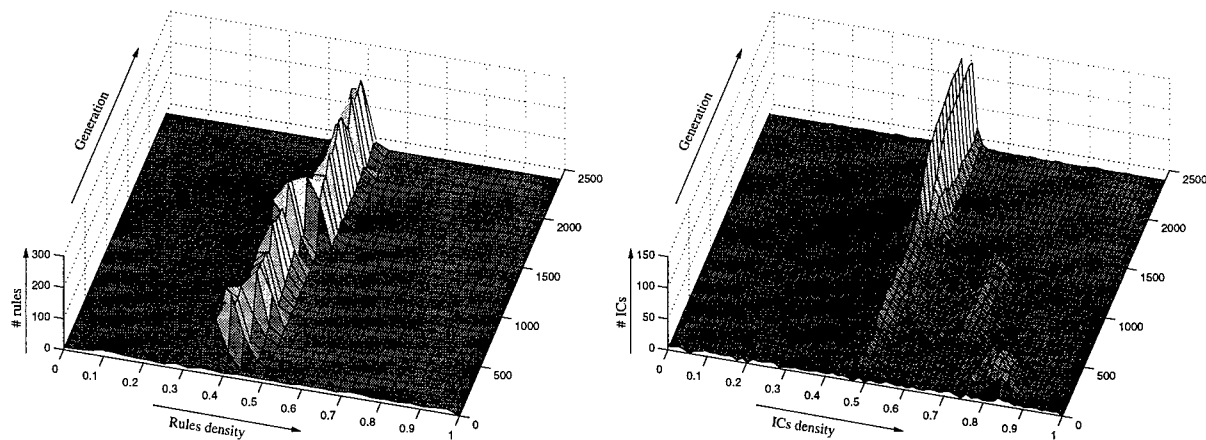


Figure 5: Coevolutionary learning between CA rules (left) and ICs (right).

- [Cliff & Miller, 1995] Cliff, D. & Miller, G. F. (1995). Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Third European Conference on Artificial Life, LNCS 929*, pp. 200–218. Springer-Verlag.
- [Das et al., 1994] Das, R., Mitchell, M., & Crutchfield, J. P. (1994). A genetic algorithm discovers particle-based computation in cellular automata. In *Parallel Problem Solving from Nature III, LNCS 866*, pp. 344–353. Springer-Verlag.
- [Epstein, 1994] Epstein, S. L. (1994). Toward an ideal trainer. *Machine Learning*, 15:251–277.
- [Hillis, 1992] Hillis, W. D. (1992). Co-evolving parasites improve simulated evolution as an optimization procedure. In Langton, C. et al. (Eds.), *Artificial Life II*, pp. 313–324. Addison Wesley.
- [Juillé & Pollack, 1996] Juillé, H. & Pollack, J. B. (1996). Co-evolving intertwined spirals. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pp. 461–468. MIT Press.
- [Land & Belew, 1995] Land, M. & Belew, R. K. (1995). No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):1548–1550.
- [Mahfoud, 1995] Mahfoud, S. W. (1995). *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign. IlliGAL Report No. 95001.
- [Mitchell et al., 1994] Mitchell, M., Crutchfield, J. P., & Hraber, P. T. (1994). Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391.
- [Mitchell et al., 1993] Mitchell, M., Hraber, P. T., & Crutchfield, J. P. (1993). Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130.
- [Paredis, 1996] Paredis, J. (1996). Coevolutionary computation. *Artificial Life*, 2(4).
- [Paredis, 1997] Paredis, J. (1997). Coevolving cellular automata: Be aware of the red queen! In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 393–400. Morgan Kaufmann.
- [Pollack et al., 1996] Pollack, J. B., Blair, A., & Land, M. (1996). Coevolution of a backgammon player. In Langton, C. (Ed.), *Proceedings of Artificial Life V*. MIT Press.
- [Rosin, 1997] Rosin, C. D. (1997). *Coevolutionary Search Among Adversaries*. PhD thesis, University of California, San Diego.
- [Rosin & Belew, 1995] Rosin, C. D. & Belew, R. K. (1995). Methods for competitive co-evolution: Finding opponents worth beating. In Eshelman, L. J. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Mateo, California. Morgan Kauffmann.
- [Schmidhuber, 1995] Schmidhuber, J. (1995). Discovering solutions with low kolmogorov complexity and high generalization capability. In Prieditis, A. & Russell, S. (Eds.), *Machine Learning: Proceedings of the twelfth International Conference*, pp. 188–196. Morgan Kaufmann.

Near-Optimal Reinforcement Learning in Polynomial Time

Michael Kearns

AT&T Labs
180 Park Avenue, Room A235
Florham Park, New Jersey 07932
mkearns@research.att.com

Satinder Singh

Department of Computer Science
University of Colorado
Boulder, Colorado 80309
baveja@cs.colorado.edu

Abstract

We present new algorithms for reinforcement learning, and prove that they have polynomial bounds on the resources required to achieve near-optimal return in general Markov decision processes. After observing that the number of actions required to approach the optimal return is lower bounded by the mixing time T of the optimal policy (in the undiscounted case) or by the horizon time T (in the discounted case), we then give algorithms requiring a number of actions and total computation time that are only polynomial in T and the number of states, for both the undiscounted and discounted cases. An interesting aspect of our algorithms is their explicit handling of the Exploration-Exploitation trade-off.

1 Introduction

In reinforcement learning, an agent interacts with an unknown environment, and attempts to choose actions that maximize its cumulative payoff (Sutton & Barto, 1998; Barto et al., 1990; Bertsekas & Tsitsiklis, 1996). The environment is typically modeled as a Markov decision process (MDP), and it is assumed that the agent does not know the parameters of this process, but has to *learn* how to act directly from experience. Thus, the reinforcement learning agent faces a fundamental trade-off between *exploitation* and *exploration* (Thrun, 1992; Sutton & Barto, 1998): should the agent exploit its cumulative experience so far, by executing the action that currently seems best, or should it execute a different action, with the hope of gaining information or experience that could lead to higher future payoffs?

Too little exploration can prevent the agent from ever converging to the optimal behavior, while too much exploration can prevent the agent from gaining near-optimal payoff in a timely fashion.

There is a large literature on reinforcement learning, which has been growing rapidly in the last decade. To the best of our knowledge, all previous results on reinforcement learning in *general* MDP's are asymptotic in nature, providing no explicit guarantees on either the number of actions or the computation time the agent requires to achieve near-optimal performance (Sutton, 1988; Watkins & Dayan, 1992; Jaakkola et al., 1994; Tsitsiklis, 1994; Gullapalli & Barto, 1994). On the other hand, finite-time results become available if one considers *restricted* classes of MDP's, if the model of learning is modified from the standard one, or if one changes the criteria for success (Saul & Singh, 1996; Fiechter, 1994; Fiechter, 1997; Schapire & Warmuth, 1994; Singh & Dayan, in press). Fiechter (1994,1997), whose results are closest in spirit to ours, considers only the discounted case, and makes the learning protocol easier by assuming the availability of a "reset" button that allows the agent to return to a fixed set of start states at any time.

Thus, despite the many interesting previous results in reinforcement learning, the literature has lacked algorithms for learning optimal behavior in *general* MDP's with provably *finite* bounds on the resources (actions and computation time) required, under the standard model of learning in which the agent wanders continuously in the unknown environment. The results presented in this paper fill this void in what is essentially the strongest possible sense.

We present new algorithms for reinforcement learning, and prove that they have *polynomial* bounds on the resources required to achieve near-optimal payoff in *general* MDP's. The bounds are polynomial in the

number of states, and also in the mixing time of the optimal policy (undiscounted case), or the horizon time $1/(1 - \gamma)$ (discounted case). One of the contributions of this work is in simply identifying the fact that finite-time convergence results *must* depend on these parameters of the underlying MDP. An interesting aspect of our algorithms is their rather explicit handling of the exploration-exploitation trade-off.

For lack of space, here we present only our results for the more difficult undiscounted case. The analogous results for the discounted case are covered in a forthcoming longer paper; interested readers can retrieve the latest version from the web page <http://www.research.att.com/~mkearns>.

2 Preliminaries and Definitions

We begin with the basic definitions for MDP's.

Definition 1 A Markov decision process (MDP) M on states $1, \dots, N$ and with actions a_1, \dots, a_k , consists of:

Transition probabilities $P_M^a(ij) \geq 0$, which for any action a , and any states i and j , specify the probability of reaching state j after executing action a from state i in M . Thus, $\sum_j P_M^a(ij) = 1$ for any state i and action a .

Payoff distributions, for each state i , with mean $R_M(i)$ (where $R_{max} \geq R_M(i) \geq 0$), and variance $Var_M(i) \leq Var_{max}$. These distributions determine the random payoff received when state i is visited.

For simplicity, we will assume that the number of actions k is a constant; it will be easily verified that if k is a parameter, the resources required by our algorithms scale polynomially with k .

Several comments regarding some benign technical assumptions that we will make on payoffs are in order here. First, it is common to assume that payoffs are actually associated with state-action pairs, rather than with states alone. Our choice of the latter is entirely for technical simplicity, and all of the results of this paper hold for the standard state-action payoffs model as well. Second, we have assumed fixed upper bounds R_{max} and Var_{max} on the means and variances of the payoff distributions; such a restriction is necessary for finite-time convergence results. Third, we have assumed that expected payoffs are always non-negative for convenience, but this is easily removed by adding the minimum expected payoff to every payoff.

If M is an MDP over states $1, \dots, N$ and with actions a_1, \dots, a_k , a **policy** in M is a mapping $\pi : \{1, \dots, N\} \rightarrow \{a_1, \dots, a_k\}$. An MDP M , combined with a policy π , yields a standard Markov process on the states, and we will say that π is *ergodic* if the Markov process resulting from π is ergodic (that is, has a well-defined stationary distribution). For the development and exposition, it will be easiest to consider MDP's for which *every* policy is ergodic, the so-called *unichain* MDP's (Puterman, 1994). Considering the unichain case simply allows us to discuss the stationary distribution of any policy without cumbersome technical details, and as it turns out, the result for unichains already forces the main technical ideas upon us. Also, note that the unichain assumption does *not* imply that every policy will eventually visit every state, or even that there exists a single policy that will do so quickly; thus, the exploration-exploitation dilemma remains with us strongly. We discuss the extension to the *multichain* case in the longer version of this paper.

If M is an MDP, then a **T -path** in M is a sequence p of $T + 1$ states (that is, T transitions) of M : $p = i_1, i_2, \dots, i_T, i_{T+1}$. The probability that p is traversed in M upon starting in state i_1 and executing policy π is $\Pr_M^\pi[p] = \prod_{k=1}^T P_M^{\pi(i_k)}(i_k i_{k+1})$. The (expected) undiscounted return **along** p in M is $U_M(p) = (1/T)(R_{i_1} + \dots + R_{i_T})$ and the T -step undiscounted return from state i is $U_M^\pi(i, T) = \sum_p \Pr_M^\pi[p] U_M(p)$, where the sum is over all T -paths p in M that start at i . We define $U_M^\pi(i) = \lim_{T \rightarrow \infty} U_M^\pi(i, T)$. Since we are in the unichain case, $U_M^\pi(i)$ is independent of i , and we will simply write U_M^π . Furthermore, we define the **optimal** T -step undiscounted return from i in M by $U_M^*(i, T) = \max_\pi \{U_M^\pi(i, T)\}$. Also, $U_M^*(i) = \lim_{T \rightarrow \infty} U_M^*(i, T)$. Finally, we observe that the maximum possible T -step return is R_{max} .

3 Mixing Times for Policies

It is easy to see that if we are seeking results about the undiscounted return of a learning algorithm after a finite number of steps, we need to take into account some notion of the *mixing times* of policies in the MDP. To put it simply, for finite-time results, there may no longer be an unambiguous notion of "the" optimal policy. There may be some policies which will eventually yield high return (for instance, by finally reaching some remote, high-payoff state), but take many steps to approach this high return, and other policies which yield lower asymptotic return but

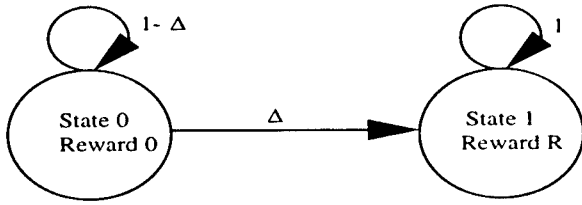


Figure 1: A simple Markov process demonstrating that finite-time convergence results must account for mixing times.

higher short-term return. Such policies are simply incomparable, and the best we could hope for is an algorithm that “competes” favorably with *any* policy, in an amount of time that is comparable to the mixing time of *that policy*.

Definition 2 Let M be an MDP, and let π be an ergodic policy in M . Then the ϵ -return mixing time of π is the smallest T such that for all $T' \geq T$, $|U_M^\pi(i, T') - U_M^\pi| \leq \epsilon$ for all i ¹.

Suppose we are simply told that there is a policy π whose asymptotic return U_M^π exceeds R in an unknown MDP M , and that the ϵ -return mixing time of π is T . In principle, a sufficiently clever learning algorithm (for instance, one that managed to discover π “quickly”) could achieve return close to $U_M^\pi - \epsilon$ in not much more than T steps. Conversely, without further assumptions on M or π , it is not reasonable to expect *any* learning algorithm to approach return U_M^π in many *fewer* than T steps. This is simply because it may take the assumed policy π itself on the order of T steps to approach its asymptotic return. For example, suppose that M has just two states and only one action (see Figure 1): state 0 with payoff 0, self-loop probability $1 - \Delta$, and probability Δ of going to state 1; and absorbing state 1 with payoff $R \gg 0$. Then for small ϵ and Δ , the ϵ -return mixing time is on the order of $1/\Delta$; but starting from state 0, it really will require on the order of $1/\Delta$ steps to reach the absorbing state 1 and start approaching the asymptotic return R . (A more formal lower bound along the lines of this argument will be given in the long version.)

¹In the long version, we relate the notion of ϵ -return mixing time to the standard notion of mixing time to stationary distributions (Puterman, 1994). The important point here is that the ϵ -return mixing time is polynomially bounded by the standard mixing time, but may in some cases be substantially smaller.

Thus, we would like a learning algorithm such that for *any* T , in a number of actions that is polynomial in T , the return of the learning algorithm is close to that achieved by the *best policy among those that mix in time T* . This motivates the following definition.

Definition 3 Let M be a Markov decision process. We define $\Pi_M^{T, \epsilon}$ to be the class of all ergodic policies π in M whose ϵ -return mixing time is at most T . We let $\text{opt}(\Pi_M^{T, \epsilon})$ denote the optimal expected asymptotic undiscounted return among all policies in $\Pi_M^{T, \epsilon}$.

Our goal in the undiscounted case will be to compete with the policies in $\Pi_M^{T, \epsilon}$ in time that is polynomial in T , $1/\epsilon$ and N . We will eventually give an algorithm that meets this goal for *every* T and ϵ *simultaneously*. An interesting special case is when $T = T^*$, where T^* is the ϵ -mixing time of the *asymptotically* optimal policy, whose asymptotic return is U^* . Then in time polynomial in T^* , $1/\epsilon$ and N , our algorithm will achieve return exceeding $U^* - \epsilon$ with high probability. It should be clear that, modulo the degree of the polynomial running time, such a result is the best that one could hope for in general MDP's. We briefly note that in the case of discounted reward, we can still hope to compete with the *asymptotically* optimal policy in time polynomial in the *horizon time*; this is discussed and achieved in the long version.

4 Main Theorem

We are now ready to describe our learning algorithm, and to state and prove our main theorem: namely, that the new algorithm will, for a general MDP, achieve near-optimal undiscounted performance in polynomial time. *For ease of exposition only*, we will first state the theorem under the assumption that the learning algorithm is given as input a “targeted” mixing time T , and the value $\text{opt}(\Pi_M^{T, \epsilon})$ of the optimal return achieved by any policy mixing within T steps. These assumptions are entirely removed in Section 4.6.

Theorem 1 (Main Theorem) Let M be a Markov decision process over N states. Recall that $\Pi_M^{T, \epsilon}$ is the class of all ergodic policies whose ϵ -return mixing time is bounded by T , and that $\text{opt}(\Pi_M^{T, \epsilon})$ is the optimal asymptotic expected undiscounted return achievable in $\Pi_M^{T, \epsilon}$. There exists an algorithm A , taking inputs ϵ, δ, N, T and $\text{opt}(\Pi_M^{T, \epsilon})$, such that if the total number of actions and computation time taken by A exceeds a polynomial in $1/\epsilon, 1/\delta, N, T$, and R_{\max} , then with probability at least $1 - \delta$, the total undiscounted return of A will exceed $\text{opt}(\Pi_M^{T, \epsilon}) - \epsilon$.

In the long version, we give a similar theorem for the discounted case (via a similar algorithm), with the horizon time playing the role of T . The criterion for success needs to be altered, however, since in the discounted case it is not possible to insist that the *actual* return achieved by the learning algorithm approach the optimal. This is due to the exponentially damped contribution of successive payoffs. Intuitively, in the discounted case it is not possible for a learning algorithm to recover from its “youthful mistakes” as it can in the undiscounted case, so we must settle for an algorithm that simply finds a near-optimal policy from its current state after a short learning period.

The remainder of this section is divided into several subsections, each describing a different and central aspect of the algorithm and proof. The full proof of the theorem is rather technical, but the underlying ideas are quite intuitive, and we sketch them first as an outline.

4.1 Overview of the Proof and Algorithm

Our algorithm will be what is commonly referred to as *indirect* or *model-based*: namely, rather than only maintaining a current policy or value function, the algorithm will actually maintain a model for the transition probabilities and the expected payoffs for some *subset* of the states of the unknown MDP M . It is important to emphasize that although the algorithm maintains a partial model of M , it may choose to *never* build a *complete* model of M , if doing so is not necessary to achieve high return.

It is easiest to imagine the algorithm as starting off by doing what we will call *balanced wandering*. By this we mean that the algorithm, upon arriving in a state it has never visited before, takes an arbitrary action from that state; but upon reaching a state it has visited before, it takes the action it has tried the fewest times from that state (breaking ties between actions randomly). At each state it visits, the algorithm maintains the obvious statistics: the average payoff received at that state so far, and for each action, the empirical distribution of next states reached (that is, the estimated transition probabilities).

A crucial notion for both the algorithm and the analysis is that of a *known state*. Intuitively, this is a state that the algorithm has visited “so many” times (and therefore, due to the balanced wandering, has tried each action from that state many times) that the transition probability and expected payoff estimates for that state are “very close” to their true values in

M . An important aspect of this definition is that it is weak enough that “so many” times is still polynomially bounded, yet strong enough to meet the simulation requirements we will outline shortly.

States are thus divided into three categories: known states, states that have been visited before, but are still unknown (due to an insufficient number of visits and therefore unreliable statistics), and states that have not even been visited once. An important observation is that we cannot do balanced wandering indefinitely before at least one state becomes known: by the Pigeonhole Principle, we will soon start to accumulate accurate statistics at some state.

Perhaps our most important definition is that of the *known-state MDP*. If S is the set of currently known states, the current known-state MDP is simply an MDP M_S that is naturally *induced* on S by the full MDP M ; briefly, all transitions in M between states in S are preserved in M_S , while all other transitions in M are “redirected” in M_S to lead to a single additional, absorbing state that intuitively represents all of the unknown and unvisited states.

Although the learning algorithm will not have direct access to M_S , by virtue of the definition of the known states, it will have an *approximation* \widehat{M}_S . The first of two central technical lemmas that we prove (Section 4.2) shows that, under the appropriate definition of known state, \widehat{M}_S will have good *simulation accuracy*: that is, the expected T -step return of any policy in \widehat{M}_S is close to its expected T -step return in M_S . (Here T is the mixing time.) Thus, at any time, \widehat{M}_S is a *partial* model of M , for that part of M that the algorithm “knows” very well.

The second central technical lemma (Section 4.3) is perhaps the most enlightening part of the analysis, and is named the “Explore or Exploit” Lemma. It formalizes a rather appealing intuition: either the optimal (T -step) policy achieves its high return by staying, with high probability, in the set S of currently known states — which, most importantly, the algorithm can *detect* and *replicate* by finding a high-return *exploitation* policy in the *partial* model \widehat{M}_S — or the optimal policy has significant probability of *leaving* S within T steps — which again the algorithm can detect and replicate by finding an *exploration* policy that quickly reaches the additional absorbing state of the partial model \widehat{M}_S .

Thus, by performing two off-line, polynomial-time computations on \widehat{M}_S (Section 4.4), the algorithm is guaranteed to either find a way to get near-optimal

return in M quickly, or to find a way to improve the statistics at an unknown or unvisited state. Again by the Pigeonhole Principle, the latter case cannot occur too many times before a new state becomes known, and thus the algorithm is always making progress. In the worst case, the algorithm will build a model of the entire MDP M , but if that does happen, the analysis guarantees that it will happen in polynomial time.

The following subsections flesh out the intuitions sketched above, providing a detailed sketch of the proof of Theorem 1; the full proofs are provided in the long version. In Section 4.6, we discuss how to remove the assumed knowledge of the optimal return and the targeted mixing time.

4.2 The Simulation Lemma

In this section, we prove the first of two key technical lemmas mentioned in the sketch of Section 4.1: namely, that if one MDP \widehat{M} is a sufficiently accurate approximation of another MDP M , then we can actually approximate the T -step return of any policy in M quite accurately by its T -step return in \widehat{M} . The important technical point is that the goodness of approximation required depends only polynomially on $1/T$, and thus the definition of known state will require only a polynomial number of visits to the state. Eventually, we will appeal to this lemma to show that we can accurately assess the return of policies in the induced known-state MDP M_S by computing their return in the algorithm's approximation \widehat{M}_S (that is, we will appeal to Lemma 2 below using the settings $M = M_S$ and $\widehat{M} = \widehat{M}_S$).

We begin with the definition of approximation we require.

Definition 4 Let M and \widehat{M} be Markov decision processes over the same state space. Then we say that \widehat{M} is an α -approximation of M if for any state i , $R_M(i) - \alpha \leq R_{\widehat{M}}(i) \leq R_M(i) + \alpha$, and for any states i and j , and any action a , $P_M^a(ij) - \alpha \leq P_{\widehat{M}}^a(ij) \leq P_M^a(ij) + \alpha$.

We now state and prove the Simulation Lemma, which says that provided \widehat{M} is sufficiently close to M in the sense just defined, the T -step return of policies in \widehat{M} and M will be similar.

Lemma 2 (Simulation Lemma) Let M be any Markov decision process over N states. Let \widehat{M} be an $O((\epsilon/(NTR_{\max}))^2)$ -approximation of M . Then for

any policy π and for any state i ,

$$U_M^\pi(i, T) - \epsilon \leq U_{\widehat{M}}^\pi(i, T) \leq U_M^\pi(i, T) + \epsilon. \quad (1)$$

Proof:(Sketch) Let \widehat{M} be an α -approximation of M , and let us fix a policy π and a start state i . Let us say that a transition from a state i' to a state j' under action a is β -small in M if $P_M^a(i'j') \leq \beta$. It is possible to bound the difference between $U_M^\pi(i, T)$ and $U_{\widehat{M}}^\pi(i, T)$ contributed by those T -paths that cross at least one β -small transition by $(\alpha + 2\beta)NTR_{\max}$ (details omitted). For the value of α stated in the theorem, our analysis chooses a value of β that yields $(\alpha + 2\beta)NTR_{\max} \leq \epsilon/4$.

Thus, for now we restrict our attention to the walks of length T that do *not* cross any β -small transition of M . It can be shown that for any T -path p that, under π , does not cross any β -small transitions of M , we have

$$(1 - \Delta)^T \Pr_M^\pi[p] \leq \Pr_{\widehat{M}}^\pi[p] \leq (1 + \Delta)^T \Pr_M^\pi[p] \quad (2)$$

where $\Delta = \alpha/\beta$. The approximation error in the payoffs yields

$$U_M(p) - \alpha \leq U_{\widehat{M}}(p) \leq U_M(p) + \alpha. \quad (3)$$

Since these inequalities hold for *any* fixed T -path that does not traverse any β -small transitions in M under π , they also hold when we take expectations over the *distributions* on such T -paths in M and \widehat{M} induced by π . Thus,

$$(1 - \Delta)^T [U_M^\pi(i, T) - \alpha] - \epsilon/4 \leq U_{\widehat{M}}^\pi(i, T) \leq (1 + \Delta)^T [U_M^\pi(i, T) + \alpha] + \epsilon/4$$

where the additive $\epsilon/4$ terms account for the contributions of the T -paths that traverse β -small transitions under π , as bounded above. The desired constraint that the outermost quantities in this chain of inequalities be separated by an additive factor of at most 2ϵ determines choices for Δ and α that yield the theorem (details omitted). \square

What role does T play in the Simulation Lemma? As we make T larger, \widehat{M} must be a better approximation of M in order to satisfy the conditions of the Simulation Lemma — but then we are guaranteed of the simulation accuracy of \widehat{M} for a larger number of steps. If we wish to “compete” with the policies in $\Pi_M^{T, \epsilon}$, then by appealing to the Simulation Lemma using T , we ensure that the *asymptotic* return in M of any policy in $\Pi_M^{T, \epsilon}$ is well approximated by its T -step return in \widehat{M} .

Thus, the Simulation Lemma essentially determines what the definition of known state should be: one that has been visited enough times to ensure (with high probability) that the estimated transition probabilities and the estimated payoffs for the state are all within $O((\epsilon/(NTR_{max}))^2)$ of their true values. A straightforward application of Chernoff bounds shows that the desired approximation will be achieved for those states from which every action has been executed at least

$$m_{known} = O(((NTR_{max})/\epsilon)^4 \text{Var}_{max} \log(1/\delta)) \quad (4)$$

times, where $\text{Var}_{max} = \max(1, \max_i[\text{Var}_M(i)])$ is the maximum of 1 and the maximum variance of the random payoffs over all states.

4.3 The “Explore or Exploit” Lemma

The Simulation Lemma indicates the degree of approximation required for sufficient simulation accuracy, and led to the definition of a known state. If we let S denote the set of known states, we now specify the straightforward way in which these known states define an induced MDP. This induced MDP has an additional “new” state, which intuitively represents all of the unknown states and transitions.

Definition 5 Let M be a Markov decision process, and let S be any subset of the states of M . The induced Markov decision process on S , denoted M_S , has states $S \cup \{s_0\}$, and transitions and payoffs defined as follows:

- For any state $i \in S$, $R_{M_S}(i) = R_M(i)$; all payoffs in M_S are deterministic (zero variance) even if the payoffs in M are stochastic.
- $R_{M_S}(s_0) = 0$.
- For any action a , $P_{M_S}^a(s_0 s_0) = 1$. Thus, s_0 is an absorbing state.
- For any states $i, j \in S$, and any action a , $P_{M_S}^a(ij) = P_M^a(ij)$. Thus, transitions in M between states in S are preserved in M_S .
- For any state $i \in S$ and any action a , $P_{M_S}^a(is_0) = \sum_{j \notin S} P_M^a(ij)$. Thus, all transitions in M that are not between states in S are redirected to s_0 in M_S .

Definition 5 describes an MDP directly induced on S by the true unknown MDP M , and as such preserves the true transition probabilities between states in S . Of course, our algorithm will only have approximations

to these transition probabilities, leading to the following obvious approximation to M_S : if we simply let \widehat{M} denote the empirical approximation to M — that is, the states of \widehat{M} are simply all the states visited so far, the transition probabilities of \widehat{M} are the observed transition frequencies, and the rewards are the observed rewards — then \widehat{M}_S is the natural approximation to M_S . Now if we let S be the set of known states, as defined by Equation (4), then the simulation accuracy of \widehat{M}_S with respect to M_S in the sense of Equation 1 follows immediately from the Simulation Lemma. Let us also observe that any return achievable in M_S (and thus approximately achievable in \widehat{M}_S) is also achievable in the “real world” M — that is, for any policy π in M , any state $i \in S$, and any T , $U_{M_S}^\pi(i, T) \leq U_M^\pi(i, T)$.

We are now at the heart of the analysis: we have identified a “part” of the unknown MDP M that the algorithm “knows” very well, in the form of the approximation \widehat{M}_S to M_S . The key lemma follows, in which we demonstrate the fact that M_S (and thus, by the Simulation Lemma, \widehat{M}_S) must always provide the algorithm with either a policy that will yield near-optimal return in the true MDP M , or a policy that will allow rapid exploration of an unknown state in M (or both).

Lemma 3 (Explore or Exploit Lemma) Let M be any Markov decision process, let S be any subset of the states of M , and let M_S be the induced Markov decision process on M . For any $i \in S$ and any T , and any $1 > \alpha > 0$, either there exists a policy π in M_S such that $U_{M_S}^\pi(i, T) \geq U_M^*(i, T) - \alpha$, or there exists a policy π in M_S such that the probability that a walk of T steps following π will terminate in s_0 exceeds α/R_{max} .

Proof: Let π be a policy in M satisfying $U_M^\pi(i, T) = U_M^*(i, T)$, and suppose that $U_{M_S}^\pi(i, T) < U_M^*(i, T) - \alpha$ (otherwise, π already witnesses the claim of the lemma). We may write

$$\begin{aligned} U_M^\pi(i, T) &= \sum_p \Pr_M^\pi[p] U_M(p) \\ &= \sum_q \Pr_M^\pi[q] U_M(q) + \sum_r \Pr_M^\pi[r] U_M(r) \end{aligned}$$

where the sums are over, respectively, all T -paths p in M , all T -paths q in M in which every state in q is in S , and all T -paths r in M in which at least one state is not in S . Keeping this interpretation of the variables p, q and r fixed, we may write $\sum_q \Pr_M^\pi[q] U_M(q) = \sum_q \Pr_{M_S}^\pi[q] U_{M_S}(q) \leq U_{M_S}^\pi(i, T)$. The equality follows from the fact that for any path q in which every state is in S , $\Pr_M^\pi[q] = \Pr_{M_S}^\pi[q]$

and $U_M(q) = U_{M_S}(q)$, and the inequality from the fact that $U_{M_S}^\pi(i, T)$ takes the sum over all T -paths in M_S , not just those that avoid the absorbing state s_0 . Thus $\sum_q \Pr_M^\pi[q] U_M(q) \leq U_M^*(i, T) - \alpha$ which implies that $\sum_r \Pr_M^\pi[r] U_M(r) \geq \alpha$. But $\sum_r \Pr_M^\pi[r] U_M(r) \leq R_{max} \sum_r \Pr_M^\pi[r]$ and so $\sum_r \Pr_M^\pi[r] \geq \alpha/R_{max}$ as desired. \square

4.4 Off-line Optimal Policy Computations

Let us take a moment to review and synthesize. The combination of the simulation accuracy of \widehat{M}_S and the Explore or Exploit Lemma establishes our basic line of argument:

- At any time, if S is the set of current known states, the T -step return of any policy π in \widehat{M}_S (approximately) lower bounds the T -step return of (any extension of) π in M .
- At any time, there must either be a policy in \widehat{M}_S whose T -step return in M is nearly optimal, or there must be a policy in \widehat{M}_S that will quickly reach the absorbing state — in which case, this same policy, executed in M , will quickly reach a state that is not currently in the known set S .

At certain points in the execution of the algorithm, we will perform T -step value iteration (which takes $O(N^2T)$ computation) off-line twice: once on \widehat{M}_S , and a second time on what we will denote \widehat{M}'_S . The MDP \widehat{M}'_S has the same transition probabilities as \widehat{M}_S , but different payoffs: in \widehat{M}'_S , the absorbing state s_0 has payoff R_{max} and all other states have payoff 0. Thus we reward exploration (as represented by visits to s_0) rather than exploitation. If $\widehat{\pi}$ is the policy returned by value iteration on \widehat{M}_S and $\widehat{\pi}'$ is the policy returned by value iteration on \widehat{M}'_S , then Lemma 3 guarantees that either the T -step return of $\widehat{\pi}$ from our current known state approaches the optimal achievable in M (which for now we are assuming we know, and can thus detect), or the probability that $\widehat{\pi}'$ reaches s_0 , and thus that the execution of $\widehat{\pi}'$ in M reaches an unknown or unvisited state in T steps with significant probability (which we can also detect). Finally, note that even though T -step value iteration produces a non-stationary policy, it is the expected payoff that is important, not whether we follow a stationary or non-stationary policy.

4.5 Putting it All Together

All of the technical pieces we need are now in place, and we now give a more detailed description of the algorithm, and sketch the remainder of the analysis. (Again, full details are provided in the long version.) We emphasize that for now we assume the algorithm is given as input a targeted mixing time T and the optimal return $opt(\Pi_M^{T,\epsilon})$ achievable in $\Pi_M^{T,\epsilon}$. In Section 4.6, we remove these assumptions.

We call the algorithm *Explicit Explore or Exploit*, or E^3 , because whenever the algorithm is not engaged in balanced wandering, it performs an explicit off-line computation on the partial model in order to find a T -step policy guaranteed to either explore or exploit.

Explicit Explore or Exploit (E^3) Algorithm:

- (Initialization) Initially, the set S of known states is empty.
- (Balanced Wandering) Any time the current state is not in S , the algorithm performs balanced wandering.
- (Discovery of New Known States) Any time a state i has been visited m_{known} times during balanced wandering, it enters the known set S , and no longer participates in balanced wandering.
- **Observation:** Clearly, after $N(m_{known} - 1) + 1$ steps of balanced wandering, by the Pigeonhole Principle *some* state becomes known. More generally, if the total number of steps of balanced wandering the algorithm has performed ever exceeds Nm_{known} , then *every* state of M is known (even if these steps of balanced wandering are not consecutive).
- (Off-line Optimizations) Upon reaching a known state $i \in S$ during balanced wandering, the algorithm performs the two off-line optimal policy computations on \widehat{M}_S and \widehat{M}'_S described in Section 4.4:
 - (Attempted Exploitation) If the resulting exploitation policy $\widehat{\pi}$ achieves return from i in \widehat{M}_S that is at least $opt(\Pi_M^{T,\epsilon}) - \epsilon/2$, the algorithm executes $\widehat{\pi}$ for the next T steps.
 - (Attempted Exploration) Otherwise, the algorithm executes the resulting exploration policy $\widehat{\pi}'$ (derived from the off-line computation on \widehat{M}'_S) for T steps in M , which by Lemma 3 is guaranteed to have probability at least $\epsilon/(2R_{max})$ of leaving the set S .
- (Balanced Wandering) Any time an attempted exploitation or attempted exploration visits a state not in S , the algorithm immediately resumes balanced wandering.

This concludes the description of the algorithm; we can now wrap up the analysis. One of the main remaining issues is our handling of the confidence parameter δ in the statement of the main theorem: Theorem 1 ensures that a certain performance guarantee is met with probability at least $1 - \delta$. There are essentially three different sources of failure for the algorithm:

- At some known state, the algorithm actually has a poor approximation to the next-state distribution for some action, and thus \widehat{M}_S does *not* have sufficiently strong simulation accuracy for M_S .
- Repeated attempted explorations fail to yield enough steps of balanced wandering to result in a new known state.
- Repeated attempted exploitations fail to result in actual return that is near $\text{opt}(\Pi_M^{T,\epsilon})$.

Our handling of the failure probability δ is to simply allocate $\delta/3$ to each of these sources of failure. The fact that we can make the probability of the first source of failure (a “bad” known state) small is handled by a standard Chernoff bound analysis applied to the definition of known states.

For the second source of failure (failed attempted explorations), a standard Chernoff bound analysis again suffices: by Lemma 3, each attempted exploration can be viewed as an independent Bernoulli trial with probability at least $\epsilon/(2R_{\max})$ of “success” (at least one step of balanced wandering). In the worst case, we must make *every* state known before we can exploit, requiring Nm_{known} steps of balanced wandering. The probability of having fewer than Nm_{known} steps of balanced wandering will be smaller than $\delta/3$ if the number of (T -step) attempted explorations is $O((R_{\max}/\epsilon)N \log(1/\delta)m_{\text{known}})$.

Finally, we do not want to simply halt upon finding a policy whose expected return is near $\text{opt}(\Pi_M^{T,\epsilon})$, but want to achieve *actual* return approaching $\text{opt}(\Pi_M^{T,\epsilon})$, which is where the third source of failure (failed attempted exploitations) enters. We have already argued that the total number of T -step attempted explorations the algorithm can perform before S contains all states of M is polynomially bounded. All other actions of the algorithm must be accounted for by T -step attempted exploitations. Each of these T -step attempted exploitations has expected return at least $\text{opt}(\Pi_M^{T,\epsilon}) - \epsilon/2$. The probability that the actual return, *restricted to just these attempted exploitations*, is less than $\text{opt}(\Pi_M^{T,\epsilon}) - 3\epsilon/4$, can be made

smaller than $\delta/3$ if the number of blocks exceeds $O((1/\epsilon)^2 \log(1/\delta))$; this is again by a standard Chernoff bound analysis. However, we also need to make sure that the return restricted to these exploitation blocks is sufficient to dominate the potentially low return of the attempted explorations. It is not difficult to show that provided the number of attempted exploitations exceeds $O(1/\epsilon)$ times the number of attempted explorations, both conditions are satisfied, for a total number of actions bounded by $O(T/\epsilon)$ times the number of attempted explorations, which is $O(NT(R_{\max}/\epsilon^2) \log(1/\delta)m_{\text{known}})$. The total computation time is thus $O(N^2T/\epsilon)$ times the number of attempted explorations, and thus bounded by

$$O(N^3T(R_{\max}/\epsilon^2) \log(1/\delta)m_{\text{known}}). \quad (5)$$

This concludes the proof of the main theorem. We remark that no serious attempt to minimize these worst-case bounds has been made; our immediate goal was to simply prove *polynomial* bounds in the most straightforward manner possible. It is likely that a practical implementation based on the algorithmic ideas given here would enjoy performance on natural problems that is considerably better than the current bounds indicate. (See Moore and Atkeson, 1993, for a related *heuristic* algorithm.)

4.6 Eliminating Knowledge of T and $\text{opt}(\Pi_M^{T,\epsilon})$

In order to simplify our presentation of the main theorem and the E^3 algorithm, we made the assumption that the learning algorithm knew the targeted mixing time T and the target optimal return $\text{opt}(\Pi_M^{T,\epsilon})$ achievable in this mixing time. In this section, we briefly outline the straightforward way in which these assumptions can be removed without changing the qualitative nature of the results. Details are in the long version of this paper.

In the absence of knowledge of $\text{opt}(\Pi_M^{T,\epsilon})$, the Explore or Exploit Lemma (Lemma 3) ensures us that it is safe to have a *bias towards exploration*. More precisely, any time we arrive in a known state i , we will *first* determine the exploration policy $\hat{\pi}'$ and compute the probability that $\hat{\pi}'$ will reach the absorbing state s_0 of \widehat{M}'_S in T steps. We can then compare this probability to the lower bound $\epsilon/(2R_{\max})$ of Lemma 3. As long as this lower bound is exceeded, we may execute $\hat{\pi}'$ in an attempt to visit a state not in S . If this lower bound is *not* exceeded, Lemma 3 guarantees that the off-line computation on \widehat{M}_S in the Attempted Exploitation step *must* result in an exploitation policy $\hat{\pi}$ that is close to optimal. We execute $\hat{\pi}$ in M and continue.

Note that this exploration-biased solution to removing knowledge of $\text{opt}(\Pi_M^{T,\epsilon})$ or $V^*(i)$ results in the algorithm always exploring all states of M that can be reached in a reasonable amount of time, before doing any exploitation. This is a simple way of removing the knowledge while keeping a polynomial-time algorithm; but we explore more practical variants of this strategy in the longer paper.

To remove the assumed knowledge of T , we observe that we already have an algorithm $A(T)$ that, given T as input, runs for $P(T)$ steps for some fixed polynomial $P(\cdot)$ and meets the desired criterion. We now propose a new algorithm A' , which does not need T as input, and simply runs A sequentially for $T = 1, 2, 3, \dots$. For any T , the amount of time A' must be run before A' has executed $A(T)$ is $\sum_{t=1}^T P(t) \leq TP(T) = P'(T)$, which is still polynomial in T . We just need to run A' for sufficiently many steps after the first $P'(T)$ steps to dominate any low-return periods that took place in those $P'(T)$ steps, similar to the analysis done for the undiscounted case towards the end of Section 4.5. We again note that this solution, while sufficient for polynomial time, is not the one we would implement in practice.

5 Conclusion

In this paper, we presented the E^3 algorithm, and showed that it achieves near-optimal undiscounted return in general MDP's in polynomial time. In the long version, we show that a slight modification of E^3 gives similar results for the discounted case, that the algorithms can deal with MDP's with terminating states in a natural way, and that they also work in multichain MDP's.

There are a number of interesting lines for further research. We are developing the basic ideas underlying E^3 into a practical algorithm, and hope to report on an implementation and experiments soon. Finding an efficient model-free version of our algorithm, and techniques for dealing with large state spaces, remain for future work.

Acknowledgements

We give warm thanks to Tom Dean, Tom Dietterich, Tommi Jaakkola, Leslie Kaelbling, Michael Littman, Lawrence Saul, Terry Sejnowski, and Rich Sutton for valuable comments. Satinder Singh was supported by NSF grant IIS-9711753.

References

- Barto, A. G., Sutton, R. S., Watkins, C. (1990). Sequential decision problems and neural networks. In *NIPS 2*, pages 686-693, Morgan Kaufmann.
- Bertsekas, D. P., Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.
- Fiechter, C. (1994). Efficient reinforcement learning. In *COLT94*, pages 88-97. ACM Press.
- Fiechter, C. (1997). Expected mistake bound model for on-line reinforcement learning. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML97)*, pages 116-124. Morgan Kaufmann.
- Gullapalli, V., Barto, A. G. (1994). Convergence of indirect adaptive asynchronous value iteration algorithms. In *NIPS 6*, pages 695-702. Morgan Kaufman.
- Jaakkola, T., Jordan, M. I., Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), 1185-1201.
- Moore, A. W., Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13(1).
- Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. New York: John Wiley & Sons.
- Saul, L., Singh, S. (1996). Learning curve bounds for Markov decision processes with undiscounted rewards. In *COLT96*.
- Schapire, R. E., Warmuth, M. K. (1994). On the worst-case analysis of temporal-difference learning algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 266-274. Morgan Kaufmann.
- Singh, S., Dayan, P. (in press). Analytical mean squared error curves for temporal difference learning. *Machine Learning*.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 9-44.
- Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.
- Thrun, S. B. (1992). The role of exploration in learning control. In White, D. A., Sofge, D. A. (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Florence, Kentucky 41022: Van Nostrand Reinhold.
- Tsitsiklis, J. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3), 185-202.
- Watkins, C. J. C. H., Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3/4), 279-292.

A Fast, Bottom-Up Decision Tree Pruning Algorithm with Near-Optimal Generalization

Michael Kearns

AT&T Labs

180 Park Avenue, Room A235
Florham Park, New Jersey 07932
mkearns@research.att.com

Yishay Mansour

Tel Aviv University

Department of Computer Science
Tel Aviv, Israel
mansour@math.tau.ac.il

Abstract

In this work, we present a new bottom-up algorithm for decision tree pruning that is very efficient (requiring only a single pass through the given tree), and prove a strong performance guarantee for the generalization error of the resulting pruned tree. We work in the typical setting in which the given tree T may have been derived from the given training sample S , and thus may badly overfit S . In this setting, we give bounds on the amount of additional generalization error that our pruning suffers compared to the *optimal* pruning of T . More generally, our results show that if there is a pruning of T with small error, and whose size is small compared to $|S|$, then our algorithm will find a pruning whose error is not much larger. This style of result has been called an *index of resolvability* result by Barron and Cover in the context of density estimation.

Our algorithm is *local* — the decision to prune a subtree is based entirely on properties of that subtree and the sample reaching it. To analyze our algorithm, we develop tools of *local uniform convergence*, a generalization of the standard notion that may prove useful in other settings.

1 Introduction

We consider the common problem of finding a good pruning of a given decision tree T on the basis of sample data S . We work in a setting in which we do *not* assume the independence of T and S . In particular, we allow for the possibility that T was in fact constructed from S , perhaps by a standard greedy, top-down process as employed in the growth phases of the C4.5 and CART algorithms [8, 3]. Our interest here is in how one should best use the data S a *second* time to find a good *subtree* of T . Note that T itself may badly overfit the data.

Our main result is a new and rather efficient pruning algorithm, and the proof of a strong performance guarantee for this algorithm (Theorems 5 and 6). Our algorithm uses the sample S to compute a subtree (pruning) of T whose *generalization* error can be related to that of the *best* pruning of T . More generally, the generalization error of our pruning is bounded by the *minimum*, over *all* prunings T' , of the generalization error $\epsilon(T')$ plus a “complexity penalty” that depends only on the size of T' . Thus, if there is a relatively small subtree of T with small error, our algorithm enjoys a strong performance guarantee. This type of guarantee is fairly common in the model selection literature, and is sometimes referred to as an *index of resolvability* guarantee [1]. (It is also similar to the types of results stated in the literature on combining “experts” [4], although the interest there is not in generalization error, but in on-line prediction. This is discussed further below.) Our algorithm is a simple, bottom-up algorithm that performs a single pass over the tree T ; hence its running time is linear in $\text{size}(T)$. The only information our algorithm needs for this bottom-up pass is, for each node in T , the depth of the node in T , the size of the subtree rooted at the node, and the number of positive and negative examples reaching the node. This information is typically available from the construction of the tree, or can be computed explicitly in time $O(|S|\text{depth}(T))$.

An important aspect of our algorithm is its *locality*. Roughly speaking, this means that the decision to prune or not prune a particular subtree during the execution is based entirely on properties of that subtree and the sample that reaches it. A number of common pruning methods behave locally in this sense. The analysis of our algorithm requires us to develop the notion of *local uniform convergence*, a generalization of the standard notion of uniform convergence, and a tool that we believe may prove useful in other settings.

2 Related Work

There are a number of previous efforts related to our results, which we only have space to discuss briefly here; more detailed comparisons will be given in the full paper. First of all, our pruning algorithm is closely related to one proposed by Mansour [7], who gave primarily experimental results, and did not give bounds on the generalization error of the resulting pruned tree.

Helmhold and Schapire [4] gave an efficient algorithm for *predicting* nearly as well as the best pruning of a given tree. However, this algorithm differs from ours in a number of important ways. First of all, it *cannot* be directly applied to the same data set that was used to derive the given tree in order to obtain a good pruning — the predictive power is only on a “fresh” or held-out data set. (A standard transformation of their algorithm can be used on the original data set, but results in a considerably less efficient algorithm, as it requires many executions of the algorithm.) Second, it does not actually find a good pruning of the given tree, but a weighted combination of prunings. However, in the on-line prediction model of learning, their result is quite strong. Here we study the typical batch model in which we may not assume independence of our tree and data set.

The use of dynamic programming for pruning was already suggested in the original book on CART [3] in order to minimize a weighted sum of the observed error and the size of the pruning. Bohanec and Bratko [2] showed that it is possible to compute in quadratic time the subtree of a given tree that minimizes the training error while obeying a specified size bound. By combining this observation with the ideas of structural risk minimization [10], it is possible to derive a polynomial-time algorithm for our setting with error guarantees quite similar to those we will give for our algorithm. However, this algorithm would be considerably less efficient than the one we shall present.

Finally, our ideas are certainly influenced by the many single-pass, bottom-up pruning heuristics in wide use in experimental machine learning, including that used by C4.5 [8]. While we do not know how to prove strong error guarantees for these heuristics, our current results provide some justification for them, and suggest specific modifications that yield fast, practical and principled methods for pruning with proven error guarantees. Combined with earlier results proving non-trivial performance guarantees for the common greedy, top-down growth heuristics in the model of boosting [5], it is fair to say that there is now a solid

theoretical basis for both the top-down and bottom-up passes of many standard decision tree learning algorithms.

3 Framework and Preliminaries

We consider decision trees over an input domain X . Each such tree has binary tests at each internal node, where each test is chosen from a class \mathcal{T} of predicates over X . We use $\text{TREES}(\mathcal{T}, d)$ to denote the class of all binary trees with tests from \mathcal{T} and at most d internal nodes, and leaves labeled with 0 or 1.

We will also need notation to identify paths in a decision tree. Thus, we use $\text{PATHS}(\mathcal{T}, \ell)$ to denote the class of all conjunctions of at most ℓ predicates from \mathcal{T} . Clearly, if v is a node in a decision tree $T \in \text{TREES}(\mathcal{T}, d)$, then we may associate with v a predicate $\text{reach}_v \in \text{PATHS}(\mathcal{T}, d)$, which is simply the conjunction of the predicates along the path from the root to v in T . Thus, for any input $x \in X$, $\text{reach}_v(x) = 1$ if and only if the path defined by x in T passes through v .

Given a node v in T , we let T_v denote the subtree of T that is rooted at v , and for any probability distribution P over X , we let P_v denote the distribution induced by P on just those x satisfying $\text{reach}_v(x) = 1$.

In our framework, there is an unknown distribution P over X and an unknown *target function* f over X . We are given a sample S of m pairs $\langle x_i, f(x_i) \rangle$, where each x_i is drawn independently according to P . We are also given a tree $T = T(S)$ that may have been built from the sample S . Now for f and T fixed, for any distribution P , we define the *generalization error* $\epsilon(T) = \epsilon_P(T) = \Pr_P[T(x) \neq f(x)]$, and also the *training error* $\hat{\epsilon}(T) = \hat{\epsilon}_S(T) = (1/m) \sum_{i=1}^m I[T(x_i) \neq f(x_i)]$, where I is the indicator function. In this notation, for any node v in T , we can define the *local generalization error* $\epsilon_v = \epsilon_{P_v}(T_v)$ and the *local training error* $\hat{\epsilon}_v = (1/|S_v|) \sum_{x \in S_v} I[T(x) \neq f(x)]$, where S_v is the set of all $x \in S$ satisfying $\text{reach}_v(x) = 1$. We will also need to refer to the local errors incurred by deleting the subtree T_v and replacing it by a leaf with the majority label of the examples reaching v . Thus, we use $\epsilon_v(\emptyset)$ to denote $\min\{\Pr_{P_v}[f(x) = 0], \Pr_{P_v}[f(x) = 1]\}$; this is exactly the error, with respect to P_v , of the optimal constant function (leaf) 0 or 1. Similarly, we will use $\hat{\epsilon}_v(\emptyset)$ to denote the quantity

$$(1/|S_v|) \min\{|\{x \in S_v : f(x) = 0\}|, |\{x \in S_v : f(x) = 1\}|\} \quad (1)$$

which is the observed local error incurred by replacing

T_v by the best leaf.

As we have mentioned in the introduction, we make no assumptions on f , and our goal is not to “learn” f in the standard sense of, say, fitting a decision tree to the data and hoping that it generalizes well. Here we limit our attention to the problem of *pruning a given* decision tree. Thus, we assume that we are given as input the sample S and a particular, fixed tree T , with the goal of finding a *pruning* of T with near-optimal generalization.

It is important to specify what we mean by a pruning of T , since allowing different pruning operations clearly can result in different classes of trees that can be obtained from T . We let $\text{PRUNINGS}(T)$ denote the class of all subtrees of T , that is, the class of all trees that can be obtained from T by specifying nodes v_1, \dots, v_k in T and then deleting from T the subtrees T_{v_1}, \dots, T_{v_k} rooted at those nodes. The allowed operation is that of deleting any subtree from the current tree; $\text{PRUNINGS}(T)$ is exactly the class of trees that can be obtained from T by any sequence of such operations. Thus, any non-empty tree in $\text{PRUNINGS}(T)$ shares the same root as T , and can be “superimposed” on T . In particular, we are not allowing operations such as the replacement of an internal node by its left or right subtree [8]. Nevertheless, the class $\text{PRUNINGS}(T)$ contains an exponential number of subtrees of T , and our goal will be find a tree in $\text{PRUNINGS}(T)$ with close to the smallest generalization error.

Let us again emphasize that we do *not* assume any “independence” between the given tree T and the sample S — indeed, the likely scenario is that T was built *using* S . Formally, we are given a pair (S, T) in which we allow $T = T(S)$. We are imagining the common scenario in which the sample S is to be used *twice* — once for top-down growth of T using a heuristic such as those used by C4.5 or CART, and now again to find a good subtree of T . If one assumes that S is a “fresh” or held-out sample (that is, drawn *separately* from the sample used to construct T), the problem becomes easier in some ways, since one can then use the observed error on S as an approximate proxy for the generalization error of any tree in $\text{PRUNINGS}(T)$. There is a trade-off that renders the two scenarios incomparable in general [6]: by using a hold-out set for the pruning phase, we gain the independence of the sample from the given tree T , but at the price of having “wasted” some potentially valuable data for the training (construction) of T ; whereas in our setting, we waste no data, but cannot exploit independence of T and S .

In the hold-out setting, a good algorithm is one that chooses the tree in $\text{PRUNINGS}(T)$ that minimizes the error on S (which can be computed in polynomial time via a dynamic programming approach [2]), and fairly general performance guarantees can be shown [6] that necessarily weaken as the hold-out set becomes a smaller fraction of the original data sample.

4 Description of the Algorithm

We begin with a detailed description of the pruning algorithm, which is given the random sample S and a tree $T = T(S)$ as input. The high-level structure of the algorithm is quite straightforward: the algorithm makes a single “bottom-up” pass through T , and decides for every node v whether to leave the subtree currently rooted at v in place (at least for the moment), or whether to delete this subtree. More precisely, imagine that we place a marker at each leaf of T , and for any node v in T , let $\text{MARKERS}(v)$ denote the set of markers in the subtree T_v rooted at v . When *all* of the markers in $\text{MARKERS}(v)$ have arrived at v , our algorithm will then (and only then) consider whether or not to delete the subtree then rooted at v ; the algorithm then passes all of these markers to its parent. Thus, the algorithm only considers pruning at a node v once it has first considered pruning at all nodes below v ; this simply formalizes the standard notion of “bottom-up” processing. Also, note that the size of the subtree rooted at v is easily computed by counting the number of markers arriving there.

Two observations are in order here. First, the algorithm considers a pruning operation only once at each node v of T , at the moment when all of $\text{MARKERS}(v)$ resides at v . Second, the subtree rooted at v when all of $\text{MARKERS}(v)$ reside at v may be *different* than T_v (the original subtree of T rooted at v), because parts of T_v may have been deleted as markers were being passed up towards v . We thus introduce the notation T_v^* to denote the subtree that is rooted at v *when all of* $\text{MARKERS}(v)$ *resides at* v . It is T_v^* that our algorithm must decide whether to prune, and T_v^* is defined by the operation of the algorithm itself. We will use T^* to denote the final pruning of T output by our algorithm.

It remains only to describe how our algorithm decides whether or not to prune T_v^* . For this we need some additional notation. We define $m_v = |S_v|$, and we let s_v denote the number of nodes in T_v^* , and ℓ_v be the depth of the node v in T . Recall that $\hat{\epsilon}_v(T_v^*)$ is the fraction of errors T_v^* makes on the local sample S_v ,

and $\hat{\epsilon}_v(\emptyset)$ is the fraction of errors the best leaf makes on S_v . Then our algorithm will replace T_v^* by this best leaf if and only if

$$\hat{\epsilon}_v(T_v^*) + \alpha(m_v, s_v, \ell_v, \delta) \geq \hat{\epsilon}_v(\emptyset) \quad (2)$$

where $\delta \in [0, 1]$ is a confidence parameter. The exact choice of $\alpha(m_v, s_v, \ell_v, \delta)$ will depend on the setting, but in all cases can be thought of as a penalty for the complexity of the subtree T_v^* . Let us first consider the case in which the class \mathcal{T} of testing functions is finite, in which case the class of possible path predicates $\text{PATHS}(\mathcal{T}, \ell_v)$ leading to v and the class of possible subtrees $\text{TREES}(\mathcal{T}, s_v)$ rooted at v are also finite. In this case, we would choose

$$\alpha(m_v, s_v, \ell_v, \delta) = c \sqrt{\frac{\log(A) + \log(B) + \log(m/\delta)}{m_v}} \quad (3)$$

for some constant specific $c > 1$ determined by the analysis, where

$$A = |\text{PATHS}(\mathcal{T}, \ell_v)| \quad (4)$$

and

$$B = |\text{TREES}(\mathcal{T}, s_v)|. \quad (5)$$

Perhaps the most natural and common special case of this finite-cardinality setting is that in which the input space X is the boolean hypercube $\{0, 1\}^n$, and the test class \mathcal{T} contains just the n single-variable tests x_i . These are the kinds of tests allowed in the vanilla C4.5 and CART packages, and since $|\text{PATHS}(\mathcal{T}, \ell)| \leq n^\ell$ and $|\text{TREES}(\mathcal{T}, s)| \leq (an)^s$ for some constant a , Equation (3) specializes to

$$\alpha(m_v, s_v, \ell_v, \delta) = c' \sqrt{\frac{(\ell_v + s_v) \log(n) + \log(m/\delta)}{m_v}} \quad (6)$$

for some specific constant $c' > 1$ determined by the analysis. To simplify the exposition and to make it more concrete, we will work with this particular choice of \mathcal{T} in most of our proofs, but specifically point out how the analysis changes for the case of infinite \mathcal{T} , where the pruning rule is given by choosing

$$\alpha(m_v, s_v, \ell_v, \delta) = c'' \sqrt{\frac{(d_{\ell_v} + d_{s_v}) \log(2m) + \log(m/\delta)}{m_v}} \quad (7)$$

for some specific constant $c'' > 1$ determined by the analysis, where d_{ℓ_v} and d_{s_v} are the VC dimensions of the classes $\text{PATHS}(\mathcal{T}, \ell_v)$ and $\text{TREES}(\mathcal{T}, s_v)$, respectively.

Let us first provide some brief intuition behind our algorithm, which will serve as motivation for the ensuing analysis as well. At each node v , our algorithm considers whether to leave the current subtree T_v^* or to delete it. The basis for this comparison must clearly make use of the sample S provided. Beyond this observation, a number of ways of comparing T_v^* to the best leaf are possible. For instance, we could simply prefer whichever of T_v^* and the best leaf makes the smaller number of mistakes on S_v . This is clearly a poor idea, since T_v^* cannot do worse than the best leaf (assuming majority labels on the leaves of T_v^*), and may do considerably better — but generalize poorly compared to the best leaf due to overfitting. Thus, it seems we should penalize T_v^* for its complexity, which is exactly the role of the additive term $\alpha(m_v, s_v, \ell_v, \delta)$ above.

One important and natural aspect of our algorithm (and many commonly used pruning methods) is the fact that the comparison between T_v^* and the best leaf is being made entirely on the basis of the *local* reduction to the observed error. That is, the comparison depends on S_v and T_v^* only, and not on all of S and T . A reasonable alternative “global” comparison might compare the observed error of the current entire tree, $\hat{\epsilon}(T^*)$, plus a penalty term that depends on *size*(T^*), with the observed error of the entire tree but with T_v^* pruned, $\hat{\epsilon}(T^* - T_v^*)$ (where $T^* - T_v^*$ is the tree after we prune at v), plus a penalty term that depends on *size*($T^* - T_v^*$). The important difference between this global algorithm and ours is that in the global algorithm, even when there is a large *absolute* difference in complexity between T_v^* and a leaf, this difference may be swamped by the fact that both are embedded in the much larger supertree T^* — that is, the difference is small *relative* to the complexity of T^* . This may cause a suboptimal insensitivity, leading to a propensity to leave large subtrees unpruned. Indeed, it is possible to construct examples in which the global approach leads to prunings strictly worse than those produced by our algorithm, and demonstrating that results as strong as we will give are not possible for the global method.

Our analysis proceeds as follows. We first need to argue that any time our algorithm chooses *not* to prune T_v^* , then (with high probability) this was in fact the “right” decision, in the sense that the current tree T^* would be degraded by deleting T_v^* . This allows us to establish that our final pruning will be a subtree of the optimal pruning, so our only source of additional error results from those subtrees of this optimal pruning that we deleted. A careful amortized analysis allows us to bound this additional error by a quantity

related to the size of the optimal pruning. This line of argument establishes a relationship between the error of our pruning and that of the optimal pruning; a slight modification of the algorithm and a more involved analysis let us make a similar comparison to *any* pruning. This extension is important for cases in which there may be a pruning whose error is only slightly worse than that of the optimal pruning, but whose size is much smaller. In such a case our bounds are much better.

5 Local Uniform Convergence

In standard uniform convergence results, we have a class of events (predicates), and we prove that the observed frequency of *any* event in the class does not differ much from its true probability. We would like to apply such results to events of the form “subtree T_v^* makes an error on x ”, but do not wish to take what is perhaps most obvious approach towards doing so. The reason is that we want to examine this event *conditioned* on the event that x reaches v , and obviously this conditioning event differs for every v . One approach would be to redefine the class of events of interest to include the conditioning events, that is, to look at events of the form “ x satisfies $reach_v(x)$ and T_v^* makes an error on x ” for all possible $reach_v(x)$ and T_v^* . It turns out that this approach would result in final bounds on our performance that are significantly worse than what we will obtain. What we really want is the rather natural notion of *local uniform convergence*: for any conditioning event c in a class C , and any event e in a class E , we would like to relate the observed frequency of e *restricted to the subsample satisfying c* to the true probability of e on the *distribution conditioned on c* ; and clearly the accuracy of this observed frequency will depend not on the overall sample size, but on the number of examples satisfying the conditioning event c . Such a relationship is given by the next two theorems, which treat the cases of finite classes and infinite classes separately.

Lemma 1 *Let C and H be finite classes of boolean functions over X , let f be a target boolean function over x , and let P be a probability distribution over X . For any $c \in C$ and $h \in H$, let $\epsilon_c(h) = \Pr_P[h(x) \neq f(x) | c(x) = 1]$, and for any labeled sample S of $f(x)$, let $\hat{\epsilon}_c(h)$ denote the fraction of points in S_c on which h errs, where $S_c = \{x \in S : c(x) = 1\}$. Then the probability that there exists a $c \in C$ and an $h \in H$*

such that

$$|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \sqrt{\frac{\log(|C|) + \log(|H|) + \log(1/\delta)}{m_c}} \quad (8)$$

is at most δ , where $m_c = |S_c|$.

Proof: Let us fix $c \in C$ and $h \in H$. For these fixed choices, we have for any value λ

$$\Pr_P[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda] = \mathbf{E}_{m_c}[\Pr_{S_c}[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda]]. \quad (9)$$

Here the expectation is over the distribution on values of m_c induced by P , and the distribution on S_c is over samples of size m_c (which is fixed inside the expectation) drawn according to P_c (the distribution P conditioned on c being 1). Since m_c is fixed, by standard Chernoff bounds we have

$$\Pr_{S_c}[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda] \leq e^{-\lambda^2 m_c} \quad (10)$$

giving the bound

$$\Pr_P[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda] \leq \mathbf{E}_{m_c}[e^{-\lambda^2 m_c}]. \quad (11)$$

If we choose

$$\lambda = \sqrt{\frac{\log(|C|) + \log(|H|) + \log(1/\delta)}{m_c}} \quad (12)$$

then $e^{-\lambda^2 m_c} = \delta/(|C||H|)$, which is a constant independent of m_c and thus can be moved outside the expectation. By appealing to the union bound ($\Pr[A \vee B] \leq \Pr[A] + \Pr[B]$), the probability that there is *some* c and h such that $|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda$ is at most $|C||H|(\delta/(|C||H|)) = \delta$, as desired. \square

Our use of Lemma 1 will be straightforward. Suppose we are considering some node v in a decision tree T at depth ℓ_v , and with a subtree T_v^* of size s_v rooted at v . Then we will appeal to the lemma choosing the conditioning class C to be the class $\text{PATHS}(\mathcal{T}, \ell_v)$, choosing H to be $\text{TREES}(\mathcal{T}, s_v)$, and choosing δ to be δ'/m^2 , where δ' is the overall confidence we desire. In this case, the local complexity penalty $\alpha(\ell_v, s_v, m_v, \delta)$ in Equation (3) and the deviation λ in Equation (12) coincide, and thus we can assert that with probability $1 - \delta'/m^2$ there is no leaf of depth ℓ_v and subtree of size s_v such that the local observed error of the subtree deviates by more than $\alpha(m_v, s_v, \ell_v, \delta)$ from the local true error. By summing over all m^2 choices for ℓ_v and s_v , we obtain an overall bound of δ' on the failure probability.

In other words, if we limit our attention to the *local* errors ϵ_v (generalization) and $\hat{\epsilon}_v$ (observed), then with high probability we can assert that they will be within an amount (namely, $\alpha(m_v, s_v, \ell_v, \delta)$) that depends only on *local* quantities: the local sample size m_v , the length ℓ_v of the path leading to v , and the size of the subtree rooted at v .

A more complicated argument is needed to prove local uniform convergence for the case of infinite classes.

Lemma 2 *Let C and H be classes of boolean functions over X , let f be a target boolean function over x , and let P be a probability distribution over X . For any $c \in C$ and $h \in H$, let $\epsilon_c(h) = \Pr_P[h(x) \neq f(x) | c(x) = 1]$, and for any labeled sample S of $f(x)$, let $\hat{\epsilon}_c(h)$ denote the fraction of points in S_c on which h errs, where $S_c = \{x \in S : c(x) = 1\}$. Then the probability that there exists a $c \in C$ and an $h \in H$ such that*

$$|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \sqrt{\frac{(d_C + d_H) \log(2m) + \log(1/\delta)}{m_c}} \quad (13)$$

is at most δ , where $m_c = |S_c|$, and d_C and d_H are the VC dimensions of C and H , respectively.

Proof:(Sketch) The proof closely follows the “two-sample trick” proof for the classical VC theorem [9], with an important variation. Intuitively, we introduce a “nested two-sample trick”, since we need to apply the idea twice — once for C , and again for H .

As in the classical proof, we define two events, but now they are “local” events. Event $A(S)$ is that in a random sample S of m examples, there exists a $c \in C$ and an $h \in H$ such that $|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda$. Event $B(S, S')$ is that in a random sample $S \cup S'$ of $2m$ examples, there exists a $c \in C$ and an $h \in H$ such that $|\hat{\epsilon}_c(h) - \hat{\epsilon}'_c(h)| \geq \lambda/2$, where $\hat{\epsilon}_c(h)$ and $\hat{\epsilon}'_c(h)$ denote the observed local error of h on S and S' , respectively.

We use the fact that

$$\Pr_S[A(S)] = \Pr_{S, S'}[A(S)] \quad (14)$$

$$= \Pr_{S, S'}[A(S) \wedge B(S, S')] / \Pr_{S, S'}[B(S, S') | A(S)] \quad (15)$$

Clearly, $\Pr_{S, S'}[A(S) \wedge B(S, S')] \leq \Pr_{S, S'}[B(S, S')]$. We also have the inequality $\Pr_{S, S'}[B(S, S') | A(S)] \geq 1/2$. Therefore, $\Pr_{S, S'}[A(S)] \leq 2\Pr_{S, S'}[B(S, S')]$, and we can concentrate on bounding the probability of event B .

Let us first consider a *fixed* set of $2m$ inputs x_1, \dots, x_{2m} . The number of possible subsets of this set induced by

taking intersections with sets in C is at most $\Phi_C(2m)$, where Φ_C is the dichotomy counting functions of classical VC analysis. Let us fix a $c \in C$, and consider the subset S_c of x_1, \dots, x_{2m} that fall in c ; let $m_c = |S_c|$. Now consider all possible labelings of S_c by the concept class H ; there are at most $\Phi_H(m_c) \leq \Phi_H(2m)$ such labelings. Let us now also fix one of these labelings, by fixing some $h \in H$.

Now both $c \in C$ and $h \in H$ are fixed. Consider splitting S_c randomly into two subsets S_c^1 and S_c^2 . For event B to hold, we need the difference between the observed errors of h on S_c^1 and S_c^2 to be at least $\lambda/2$. It can be shown that this will occur with probability at most $e^{-\lambda^2 m_c / 12}$, where the probability is taken only over the random partitioning of S_c . Now if we choose

$$\lambda = \sqrt{\frac{(d_C + d_H) \log(2m) + \log(1/\delta)}{m_c}} \quad (16)$$

then $e^{-\lambda^2 m_c / 12} = (1/(2m)^{d_C})(1/(2m)^{d_H})\delta$, which is independent of m_c . We can then bound the probability that this event occurs for *some* c and h by summing this bound over all possible subsets S_c , and all possible labelings of S_c by functions in H , giving a bound of $\Phi_C(2m)\Phi_H(2m)(1/m^{d_C})(1/m^{d_H})\delta$. Using the fact that $\Phi_C(m) \leq m^{d_C}$ and $\Phi_H(m) \leq m^{d_H}$ yields an overall bound of δ , as desired. \square

6 Analysis of the Pruning Algorithm

In this section, we apply the tools of local uniform convergence to analyze the pruning algorithm given in Section 4. As mentioned earlier, for simplicity in exposition, we will limit our attention to the common case in which X is the boolean hypercube $\{0, 1\}^n$ and the class \mathcal{T} of allowed node tests is just the input variables x_i , in which case the pruning rule used by our bottom-up algorithm is that given by Equation (6). However, it should be clear how the analysis easily generalizes to the more general algorithms given by Equations (3) and (7).

We shall first give an analysis that compares the generalization error of the pruning T^* produced by our algorithm from S and T to the generalization error of T_{opt} , the pruning of T that minimizes the generalization error. Recall that we use T_v^* to denote the subtree that is rooted at node v of T at the time our algorithm decides whether or not to prune at v , which may be a subtree of T_v due to prunings that have already taken place below v .

We will show that $\epsilon(T^*)$ is larger than $\epsilon_{opt} = \epsilon(T_{opt})$

by an amount that can be bounded by a function of the size s_{opt} and depth ℓ_{opt} of T_{opt} . Thus, if there is a reasonably small subtree of T with small generalization error, our algorithm will produce a pruning with small generalization error as well. In Section 7, we will improve our analysis to compare the error of T^* to that of *any* pruning, and provide a discussion of situations in which this result may be considerably more powerful than our initial comparison to T_{opt} alone.

For the analysis, it will be convenient to introduce the notation

$$r_v = (\ell_v + s_v) \log(n) + \log(m/\delta) \quad (17)$$

for any node v , where ℓ_v is the depth of v in T , and s_v is the size of T_v^* . In this notation, the penalty $\alpha(m_v, s_v, \ell_v, \delta)$ given by Equation (6) is simply a constant (that we ignore in the analysis for ease of exposition) times $\sqrt{r_v/m_v}$. (We assume that $\sqrt{r_v/m_v} \leq 1$, since a penalty which is larger than 1 can be modified to a penalty of 1 without changing the results.)

Lemma 3 *With probability at least $1 - \delta$ over the draw of the input sample S , T^* is a subtree of T_{opt} .*

Proof: Consider any node v that is a leaf in T_{opt} . It suffices to argue that our algorithm would choose to prune T_v^* , the subtree that remains at v when our algorithm reaches v . By Equation (6), our algorithm would *fail* to prune T_v^* only if $\hat{\epsilon}_v(\emptyset)$ exceeded $\hat{\epsilon}_v(T_v^*)$ by at least the amount $\alpha(m_v, s_v, \ell_v, \delta)$, in which case Lemma 1 ensures that $\epsilon_v(T_v^*) < \epsilon_v(\emptyset)$ with high probability. In other words, if our algorithm fails to prune T_v^* , then T_{opt} would have smaller generalization error by including T_v^* rather than making v a leaf. This contradicts the optimality of T_{opt} . \square

Lemma 3 means that the only source of additional error of T^* compared to T_{opt} is through overpruning, not underpruning. Thus, for the purposes of our analysis, we can imagine that our algorithm is actually run on T_{opt} rather than the original input tree T (that is, the algorithm is initialized starting at the leaves of T_{opt} , since we know that the algorithm will prune everything below this frontier).

Let $V = \{v_1, \dots, v_t\}$ be the sequence of nodes in T_{opt} at which the algorithm chooses to prune the subtree $T_{v_i}^*$ rather than to leave it in place; note that $t \leq s_{opt}$. Then we may express the additional generalization error $\epsilon(T^*) - \epsilon_{opt}$ as

$$\epsilon(T^*) - \epsilon_{opt} = \sum_{i=1}^t (\epsilon_{v_i}(\emptyset) - \epsilon_{v_i}(T_{v_i}^*)) p_{v_i} \quad (18)$$

where p_{v_i} is the probability under the input distribution P of reaching node v_i , that is, the probability of satisfying the path predicate $reach_{v_i}$. Each term in the summation of Equation (18) simply gives the change to the *global* error incurred by pruning $T_{v_i}^*$, expressed in terms of the *local* errors. Clearly the additional error of T^* is the sum of all such changes.

Now we may write

$$\begin{aligned} \epsilon(T^*) - \epsilon_{opt} &\leq \sum_{i=1}^t (|\epsilon_{v_i}(\emptyset) - \hat{\epsilon}_{v_i}(\emptyset)| + |\hat{\epsilon}_{v_i}(\emptyset) - \hat{\epsilon}_{v_i}(T_{v_i}^*)| \\ &\quad + |\hat{\epsilon}_{v_i}(T_{v_i}^*) - \epsilon_{v_i}(T_{v_i}^*)|) p_{v_i} \end{aligned} \quad (19)$$

$$\begin{aligned} &\leq \sum_{i=1}^t \left(\sqrt{\frac{(\ell_{v_i} + 1) \log(n) + \log(m/\delta)}{m_{v_i}}} \right. \\ &\quad \left. + \alpha(m_{v_i}, s_{v_i}, \ell_{v_i}, \delta) + \sqrt{\frac{(\ell_{v_i} + s_{v_i}) \log(n) + \log(m/\delta)}{m_{v_i}}} \right) p_{v_i} \end{aligned} \quad (20)$$

$$\leq 4 \sum_{i=1}^t \left(\sqrt{\frac{r_{v_i}}{m_{v_i}}} \right) p_{v_i}. \quad (21)$$

The first inequality comes from the triangle inequality. The second inequality uses two invocations of Lemma 1, and the fact that our algorithm directly compares $\hat{\epsilon}_{v_i}(\emptyset)$ and $\hat{\epsilon}_{v_i}(T_{v_i}^*)$, and prunes only when they differ by less than $\alpha(m_{v_i}, s_{v_i}, \ell_{v_i}, \delta)$.

Thus, we would like to bound the sum $\Delta = \sum_{i=1}^t (\sqrt{r_{v_i}/m_{v_i}}) p_{v_i}$. The leverage we will eventually use is the fact that $\sum_{i=1}^t r_{v_i}$ can be bounded by quantities involving only the tree T_{opt} , since all of the $T_{v_i}^*$ are disjoint subtrees of T_{opt} . First it will be convenient to break this sum into two sums — one involving just those terms for which p_{v_i} is “small”, and the other involving just those terms for which p_{v_i} is “large”. The advantage is that for the large p_{v_i} , we can relate p_{v_i} to its empirical estimate $\hat{p}_{v_i} = m_{v_i}/m$, as evidenced by the following lemma.

Lemma 4 *The probability, over the sample S , that there exists a node $v_i \in V$ such that $p_{v_i} > 12 \log(t/\delta)/m$ but $p_{v_i} \geq 2\hat{p}_{v_i}$, is at most δ .*

Proof: We will use the relative Chernoff bound

$$\Pr[\hat{p}_{v_i} < (1 - \gamma)p_{v_i}] \leq e^{-m\gamma^2/3} \quad (22)$$

which holds for any fixed v_i . By taking $\gamma = 1/2$ and applying the union bound, we obtain

$$\Pr[\exists v_i \in V : p_{v_i} \geq 2\hat{p}_{v_i}] \leq te^{-p_{v_i}m/12}. \quad (23)$$

Now we can use the assumed lower bound on p_{v_i} to bound the probability of the event by δ . \square

Let V' be the subset of V for which the lower bound $p_{v_i} > 12 \log(t/\delta)/m$ holds. We divide the sum that describes Δ into two parts:

$$\Delta = \sum_{v_i \in V - V'} \left(\sqrt{r_{v_i}/m_{v_i}} \right) p_{v_i} + \sum_{v_i \in V'} \left(\sqrt{r_{v_i}/m_{v_i}} \right) p_{v_i} \quad (24)$$

The first sum is bounded by $12 \log(s_{opt}/\delta) s_{opt}/m$, since $\sqrt{r_{v_i}/m_{v_i}}$ is at most 1, and $t \leq s_{opt}$.

For the second sum, we perform a maximization. By Lemma 4, with high probability we have that for every $v_i \in V'$, $p_{v_i} < 2\hat{p}_{v_i} = 2m_{v_i}/m$. Thus, with high probability we have

$$\sum_{v_i \in V'} \sqrt{\frac{r_{v_i}}{m_{v_i}}} p_{v_i} < \sum_{v_i \in V'} \sqrt{\frac{r_{v_i}}{m_{v_i}}} \left(2 \frac{m_{v_i}}{m} \right) \quad (25)$$

$$= \frac{2}{m} \sum_{v_i \in V'} \sqrt{r_{v_i}} \sqrt{m_{v_i}} \quad (26)$$

$$\leq \frac{2}{m} \sqrt{\left(\sum_{v_i \in V'} r_{v_i} \right) \left(\sum_{v_i \in V'} m_{v_i} \right)} \quad (27)$$

To bound this last expression, we first bound $\sum_{v_i \in V'} r_{v_i}$. Recall that

$$r_{v_i} = (\ell_{v_i} + s_{v_i}) \log(n) + \log(m/\delta). \quad (28)$$

Since for any $v_i \in V'$, we have $\ell_{v_i} \leq \ell_{opt}$, we have that $\sum_{v_i \in V'} \ell_{v_i} \leq s_{opt} \ell_{opt}$, since $|V'| \leq t \leq s_{opt}$. Since the subtrees $T_{v_i}^*$ that we prune are disjoint and subsets of the optimal subtree T_{opt} , we have $\sum_{v_i \in V'} s_{v_i} \leq s_{opt}$. Thus

$$\sum_{v_i \in V'} r_i \leq s_{opt}((1 + \ell_{opt}) \log(n) + \log(m/\delta)). \quad (29)$$

To bound $\sum_{v_i \in V'} m_{v_i}$ in Equation (27), we observe that since the sets of examples that reach different nodes at the same depth in the tree are disjoint, we have $\sum_{v_i \in V'} m_{v_i} \leq m \ell_{opt}$. Thus, with probability $1 - \delta$, we obtain an overall bound

$$\Delta < 12 \log(s_{opt}/\delta) \frac{s_{opt}}{m} + \frac{2}{m} \sqrt{s_{opt}((1 + \ell_{opt}) \log(n) + \log(m/\delta))(m \ell_{opt})} \quad (30)$$

$$= O\left(\left(\log(s_{opt}/\delta) + \ell_{opt} \sqrt{\log(n) + \log(m/\delta)}\right) \times \sqrt{\frac{s_{opt}}{m}}\right) \quad (31)$$

This gives the first of our main results.

Theorem 5 *Let S be a random sample of size m drawn according an unknown target function and input distribution. Let $T = T(S)$ be any decision tree, and let T^* denote the subtree of T output by our pruning algorithm on inputs S and T . Let ϵ_{opt} denote the smallest generalization error among all subtrees of T , and let s_{opt} and ℓ_{opt} denote the size and depth of the subtree achieving ϵ_{opt} . Then with probability $1 - \delta$ over S ,*

$$\begin{aligned} \epsilon(T^*) - \epsilon_{opt} \\ = O\left(\left(\log(s_{opt}/\delta) + \ell_{opt} \sqrt{\log(n) + \log(m/\delta)}\right) \times \sqrt{s_{opt}/m}\right) \end{aligned} \quad (32)$$

7 An Index of Resolvability Result

Roughly speaking, Theorem 5 ensures that the true error of the pruning found by our algorithm will be larger than that of the best possible pruning by an amount that is not much worse than $\sqrt{s_{opt}/m}$ (ignoring logarithmic and depth factors for simplicity). How good is this? Since we assume that T itself (and therefore, all subtrees of T) may have been constructed from the sample S , standard model selection analyses [10] indicate that ϵ_{opt} may be larger than the error of the best decision tree approximation to the target function by an amount growing like $\sqrt{s_{opt}/m}$. (Recall that ϵ_{opt} is only the error of the optimal *subtree* of T — there may be other trees which are not subtrees of T with error less than ϵ_{opt} , especially if T was constructed by a greedy top-down heuristic.) Thus, if we only compare our error to that of T_{opt} , we are effectively only paying an additional penalty of the same order that T_{opt} pays. If s_{opt} is small compared to m — that is, the optimal subtree of T is small — then this is quite good indeed.

But a stronger result is possible and desirable. Suppose that T_{opt} is not particularly small, but that there is a much smaller subtree T' whose error is not much worse than ϵ_{opt} . In such a case, we would rather claim that our error is close to that of T' , with a penalty that goes only like $\sqrt{s'/m}$. This was the *index of resolvability* criterion for model selection first examined for density estimation by Barron and Cover [1], and we now generalize our main result to this setting.

Theorem 6 *Let S be a random sample of size m drawn according an unknown target function and input*

distribution. Let $T = T(S)$ be any decision tree, and let T^* denote the subtree of T output by our pruning algorithm on inputs S and T . Then with probability $1 - \delta$ over S ,

$$\begin{aligned} & \epsilon(T^*) \\ & \leq \min_{T'} \left\{ \epsilon(T') + O \left(\left(\log(s_{\text{eff}}(T')/\delta) + \right. \right. \right. \\ & \quad \left. \left. \left. \ell_{\text{eff}}(T') \sqrt{\log(n) + \log(m/\delta)} \right) \sqrt{s_{\text{eff}}(T')/m} \right) \right\}. \end{aligned} \quad (33)$$

Here the min is taken over all subtrees T' of T , and we define the "effective" size

$$s_{\text{eff}}(T') = s' + 2m(\epsilon(T') - \epsilon_{\text{opt}}) + 6s' \log(s'/\delta) \quad (34)$$

and the "effective" depth $\ell_{\text{eff}}(T') = \min\{\ell_{\text{opt}}, s'\}$, where s' and ℓ' are the size and depth of T' , ϵ_{opt} denotes the smallest generalization error among all subtrees of T , and ℓ_{opt} denotes the depth of the subtree achieving ϵ_{opt} .

The proof is omitted due to space considerations, but the main difference from the proof of Theorem 5 is that our pruning is no longer a subtree of the pruning T' to which it is being compared. This requires a slight modification of the pruning penalty $\alpha(m_v, s_v, \ell_v, \delta)$, and the analysis bounding the sum of the sizes of the pruned subtrees becomes more involved.

Again ignoring logarithmic and depth factors for simplicity, Theorem 6 compares the error of our pruning *simultaneously* to all prunings T' . Our additional error goes roughly like $\sqrt{s_{\text{eff}}(T')/m}$. In Equation (34), if s' is small compared to m and $\epsilon(T')$ is not much larger than ϵ_{opt} , then the bound shows that our error will compare well to ϵ_{opt} — even though the tree T' achieving the min may not be T_{opt} . This is the power of the guarantee provided by index of resolvability results.

Acknowledgements

Thanks to Rob Schapire for discussions of on-line methods for predicting nearly as well as the best pruning. Y. Mansour was supported in part by a grant from the Israel Science Foundation.

References

- [1] Andrew R. Barron and Thomas M. Cover. Minimum Complexity Density Estimation. *IEEE Transactions on Information Theory*, Vol. 37, No. 4, pages 1034 – 1054, 1991.
- [2] Marco Bohanec and Ivan Bratko. Trading Accuracy for simplicity in Decision Trees. *Machine Learning*, Vol. 15, pages 223 – 250, 1994.
- [3] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone. Classification and Regression Trees. Wadsworth International Group, 1984.
- [4] David P. Helmbold and Robert E. Schapire. Predicting Nearly as Well as the Best Pruning of a Decision Tree. *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, ACM Press, pages 61 – 68, 1995.
- [5] Michael Kearns and Yishay Mansour. On the Boosting Ability of Top-Down Decision Tree Learning Algorithms. *Proceedings of the 23th Annual ACM Symposium on the Theory of Computing*, ACM Press, pages 459–468, 1996.
- [6] M. Kearns, Y. Mansour, A. Ng, D. Ron. An Experimental and Theoretical Comparison of Model Selection Methods. *Machine Learning*, 27(1):7–50, 1997.
- [7] Yishay Mansour. Pessimistic Decision Tree Pruning Based on Tree Size. *Proceedings of the Fourteenth International Conference on Machine Learning*, Morgan Kaufmann, pages 195 – 201, 1997.
- [8] J. Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.
- [9] V.N. Vapnik and A. Ya. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Theory of Probability and its Applications*, XVI(2):264–280, 1971.
- [10] V.N. Vapnik. Estimation of Dependences Based on Empirical Data. Springer-Verlag, 1982.

An Analysis of Actor/Critic Algorithms using Eligibility Traces: Reinforcement Learning with Imperfect Value Functions

Hajime Kimura*

Tokyo Institute of Technology
gen@fe.dis.titech.ac.jp

Shigenobu Kobayashi

Tokyo Institute of Technology
kobayasi@dis.titech.ac.jp

Abstract

We present an analysis of actor/critic algorithms, in which the actor updates its policy using eligibility traces of the policy parameters. Most of the theoretical results for eligibility traces have been for only critic's value iteration algorithms. This paper investigates what the actor's eligibility trace does. The results show that the algorithm is an extension of Williams' REINFORCE algorithms for infinite horizon reinforcement tasks, and then the critic provides an appropriate reinforcement baseline for the actor. Thanks to the actor's eligibility trace, the actor improves its policy by using a gradient of actual return, not by using a gradient of the estimated return in the critic. It enables the agent to learn a fairly good policy under the condition that the approximated value function in the critic is hopelessly inaccurate for conventional actor/critic algorithms. Also, if an accurate value function is estimated by the critic, the actor's learning is dramatically accelerated in our test cases. The behavior of the algorithm is demonstrated through simulations of a linear quadratic control problem and a pole balancing problem.

1 Introduction

Actor/critic architecture is an adaptive version of policy iteration [Kaelbling et al. 96]. In general, policy iteration alternates two phases: a policy evaluation phase and a policy improvement phase. The actor implements a *stochastic policy* that maps from a representation of a state to a probability distribution over

actions. The critic attempts to estimate the evaluation function for the current policy. The actor improves its control policy using critic's *temporal difference (TD)* as an effective reinforcement. In many cases, the policy improvement is executed concurrently with the policy evaluation, because it is not feasible to wait for the policy evaluation to converge.

The actor/critic algorithms have been successfully applied to a variety of delayed reinforcement tasks; ASE/ACE architecture for a pole balancing [Barto et al. 83] [Gullapalli 92], RFALCON for a pole balancing and for control of a ball-beam system [Lin et al. 96], a cart-pole swing-up task [Doya 96]. Although convergence proofs for the actor/critic algorithms (e.g. [Williams et al. 90] and [Gullapalli 92]) are less than value-iteration based algorithms such as Q-learning [Watkins et al. 92], the actor/critic algorithms have the following practical advantages.

- It is easy to implement multidimensional continuous action, that is often mixed with discrete action [Gullapalli 92]. Because the actor selects action by its stochastic policy, therefore problems of action selection like as Q-learning does not exist. The Q-learning needs to estimate returns for all state-action pairs, but the critic would estimate only the return of each state.
- Memory-less stochastic policies can be considerably better than memory-less deterministic policies in the case of partially observable Markov decision processes (POMDPs) [Singh 94] [Jaakkola 94] or multi-player games [Littman 94].
- It is easy to incorporate an expert's knowledge into the learning system by applying conventional supervised learning techniques to the actor [Clouse et al. 92].

Eligibility traces are a fundamental mechanism that has been widely used to handle delayed reward [Singh 96]. Also the traces are often used to overcome non-Markovian effects [Sutton 95],

* Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, 4259 Nagatsuta Midori-ku Yokohama 226-8502 JAPAN.

[Pendrith et al. 96]. In Barto, Sutton and Anderson's ASE/ACE architecture, both the critic and the actor make use of the eligibility trace. Theoretical results of eligibility traces in the context of TD(λ) [Sutton 88] have been obtained. But, in actor/critic algorithms, the effect of the actor's trace has not been investigated. This paper presents an analysis of an actor/critic algorithm, in which the actor improves its policy using eligibility traces of the policy parameters. This may be the first analysis of the actor's eligibility traces.

2 Discounted Reward Criteria

At each discrete time t , the agent observes x_t containing information about its current state, select action a_t , and then receives an instantaneous reward r_t resulting from state transition in the environment. In general, the reward and the next state may be random, but their probability distributions are assumed to depend only on x_t and a_t in Markov decision processes (MDPs), in which many reinforcement learning algorithms are studied. The objective of reinforcement learning is to construct a policy that maximizes the agent's performance. A natural performance measure for infinite horizon tasks is the cumulative discounted reward:

$$V_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad (1)$$

where the discount factor, $0 \leq \gamma < 1$ specifies the importance of future rewards. V_t is called the *actual return*, that specifies how good the reward sequence after time t is. By this notation, the goal of the learning is to maximize the *expected return*. In MDPs, the expected return can be defined for all states as:

$$V^{\pi}(x) = E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_k | x_0 = x \right], \quad (2)$$

where E_{π} denotes the expectation assuming the agent always uses stationary policy π . $V^{\pi}(x)$ is called the *value function*, that specifies how good the given state x is. In MDPs, the goal of the learning is to find an optimal policy that maximizes the value of each state x defined by Equation 2. Although similar value functions can be given in POMDPs, difficulties to define the optimum have pointed out in [Singh 94].

3 Actor/Critic Algorithms

Figure 1 and 2 give an overview of actor/critic algorithms [Sutton 90] [Crites et al. 94]. There are many ways to implement the policy and its updating scheme in the actor. The algorithms for the critic are mostly TD methods. We should notice the following two points; one is the actor implements stochastic policy,

the other is the actor improves its policy using TD-error. This paper especially investigates an algorithm for the actor.

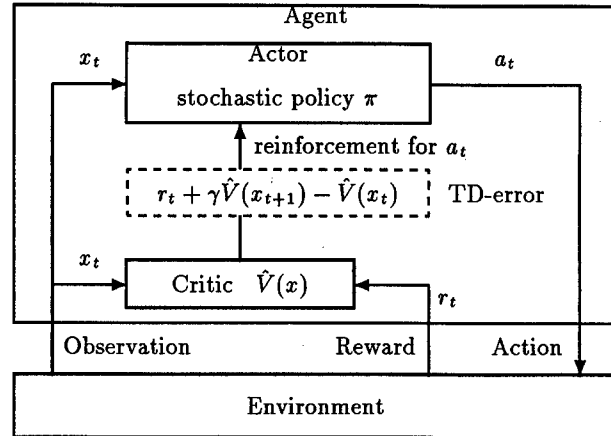


Figure 1: A generic actor/critic framework.

1. The agent observes x_t in the environment, and the actor executes action a_t according to the current stochastic policy π .
2. The critic receives the immediate reward r_t , and then observes the resulting next state x_{t+1} . The critic provides TD error as an useful reinforcement feedback to the actor, according to

$$(\text{TD-error}) = [r_t + \gamma \hat{V}(x_{t+1})] - \hat{V}(x_t),$$

where $0 \leq \gamma < 1$ is the discount factor, $\hat{V}(x)$ is an estimated value function by the critic.

3. The actor updates the stochastic policy using the TD-error. If (TD-error) > 0 , action a_t performed relatively good and its probability should be increased. If (TD-error) < 0 , action a_t performed relatively poorly and its probability should be decreased.
4. The critic updates estimated value function $\hat{V}(x)$ according to TD methods. e.g., TD(0) algorithm adjusts $\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha (\text{TD-error})$, where α is the learning rate.
5. Go to step 1.

Figure 2: Main loop of the generic actor/critic algorithm.

4 Adding Eligibility Trace to the Actor

4.1 Function Approximation for Stochastic Policies

In this paper, $\pi(a, W, x)$ denotes probability of selecting action a under the policy π in the observation x . The $\pi(a, W, X)$ is taken to be a probability density function when the set of possible action is continuous. The policy is represented by a parametric function approximator using the internal variable vector W . The agent can improve the policy π by modifying W . For example, W corresponds to synaptic weights where the action selecting probability is represented by neural networks, or W means weight of rules in classifier systems. The advantage of using the notation of the parametric function $\pi()$ is that computational restriction and mechanisms of the agent can be specified simply by a form of the function, and then we can provide a sound theory of learning algorithms for arbitrary types of the actor.

4.2 Details of the Algorithm

Figure 3 specifies the actor/critic algorithm that uses the eligibility trace in the actor. The ASE/ACE system configured for pole-balancing [Barto et al. 83] is just an instance of this algorithm. The actor's eligibility in step 3 is the same variable defined in Williams' REINFORCE algorithms [Williams 92]. The eligibility $e_i(t)$ specifies a correlation between the associated policy parameter w_i and the executed action a_t . The eligibility trace $D_i(t)$ is a discounted running average of eligibility. It accumulates the agent's history. When a positive reinforcement is given, the actor updates W so that the probability of actions recorded in the history is increased. It means the TD-error at the time t affects not only the action a_t but also a_{t-1}, a_{t-2}, \dots . At first glance, this idea is senseless for improving the policy, but it has very interesting features given in detail later. Note that the algorithm shown in Figure 3 is identical to a stochastic gradient ascent for discounted reward [Kimura et al. 97] when the actor's discount factor $\beta = \gamma$ and the $\hat{V}(x)$ in the critic equals a constant b for all observations.

The actor requires a memory to implement W for the policy and to implement D_i for the eligibility trace. The amount of the memory for D_i is equal to W 's.

4.3 An Analysis of the Algorithm

Assume that the actor's discount factor β equals γ , and for all $t < 0$, $D_i(t) = 0$, then the algorithm shown

1. The agent observes x_t , and the actor executes action a_t with probability $\pi(a_t, W, x_t)$.
2. The critic receives the immediate reward r_t , and then observes the resulting next state x_{t+1} . The critic provides TD error to the actor according to

$$(\text{TD-error}) = [r_t + \gamma \hat{V}(x_{t+1})] - \hat{V}(x_t), \quad (3)$$

where $0 \leq \gamma < 1$ is the discount factor, $\hat{V}(x)$ is an estimated value function by the critic.

3. The actor updates the stochastic policy using the TD-error according to:

$$\begin{aligned} \text{Eligibility:} \quad e_i(t) &= \frac{\partial}{\partial w_i} \ln(\pi(a_t, W, x_t)) \\ \text{Eligibility Trace:} \quad D_i(t) &= e_i(t) + \beta D_i(t-1), \\ \Delta w_i(t) &= (\text{TD-error}) D_i(t) \\ W &\leftarrow W + \alpha_p \Delta W(t), \end{aligned}$$

where w_i denotes the i^{th} component of W , e_i and D_i are the associated eligibility and eligibility trace respectively, β ($0 \leq \beta < 1$) is a discount factor for the eligibility trace, α_p is the learning rate for the actor.

4. The critic updates estimated value function $\hat{V}(x)$ according to TD methods. e.g., TD(0) algorithm adjusts $\hat{V}(x) \leftarrow \hat{V}(x) + \alpha (\text{TD-error})$, where α is the learning rate.

5. Go to step 1.

Figure 3: The actor/critic algorithm adding the eligibility trace to the actor.

in Figure 3 updates the policy parameters as:

$$\begin{aligned} \sum_{t=0}^{\infty} \Delta w_i(t) &= \sum_{t=0}^{\infty} (r_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t)) D_i(t) \\ &= \sum_{t=0}^{\infty} (r_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t)) \left(\sum_{\tau=0}^t \gamma^{t-\tau} e_i(\tau) \right) \\ &= \sum_{t=0}^{\infty} e_i(t) \left(\sum_{\tau=t}^{\infty} \gamma^{\tau-t} (r_{\tau} + \gamma \hat{V}(x_{\tau+1}) - \hat{V}(x_{\tau})) \right) \\ &= \sum_{t=0}^{\infty} e_i(t) \left(\left(\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau} \right) - \hat{V}(x_t) \right) \quad (4) \\ &= \sum_{t=0}^{\infty} e_i(t) (V_t - \hat{V}(x_t)) \quad (5) \end{aligned}$$

Equation 5 is given by Equation 1 and 4. Here we assume that the statistics of the random variable V_t depends only on the current policy parameter. It means $E\{V_t\}$ is a deterministic function of W , where E de-

notes the expectation operator. This assumption may be right if the policy is converged to an equilibrium point. The critic's estimation $\hat{V}(x_t)$ is obviously independent of the action at the time t . From the theory of Williams' REINFORCE algorithm [Williams 92], the value V_t and $\hat{V}(x_t)$ in Equation 5 can be seen as a reinforcement signal and a reinforcement baseline respectively, then we have $E\{e_i(t)(V_t - \hat{V}(x_t))\} = (\partial/\partial w_i)E\{V_t\}$. It says that the algorithm updates policy parameters statistically in a direction for increasing the actual return V_t , not in a direction of a gradient of estimated value function in the critic. Also It can be seen as an extension of reinforcement comparison methods [Sutton et al. 98], then $\hat{V}(x_t)$ corresponds to the reference reward.

From the above analysis and Figure 3, we can explain what the actor's eligibility trace does. At the time t , the algorithm reinforces a_t using TD error $r_t + \hat{V}(x_{t+1}) - \hat{V}(x_t)$ as a temporary expedient, thereafter the actor's eligibility trace replaces $\hat{V}(x_{t+1})$ with the actual return $(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots)$ in order.

The critic does not affect the direction of the average update vector, because the critic works as a reinforcement baseline. Therefore, the actor can improve its policy, whether the critic is able to learn the value function or not. If the critic approximates the value function well, the actor's learning would be accelerated.

The above results are under the special condition $\beta = \gamma$. If $\beta = 0$, the actor updates W in the direction of the gradient of the approximated value function in the critic. The β ($0 < \beta < \gamma$) interpolates between the above two limiting cases. The characteristics of the β are similar to the λ in TD(λ) [Sutton 88] and Q(λ)-learning [Peng et al. 94].

5 Preliminary Experiments

This section demonstrates the performance of the algorithm applying to a simple linear control problem.

5.1 A Linear Quadratic Regulator (LQR)

The following linear control problem can serve as a benchmark of delayed reinforcement tasks [Baird 94]. At a given discrete-time t , the state of the environment is the real value x_t . The agent chooses a control action a_t that is also real value. The dynamics of the environment is:

$$x_{t+1} = x_t + a_t + \text{noise}, \quad (6)$$

where the *noise* is the normal distribution that follows the standard deviation $\sigma_{\text{noise}} = 0.5$. The immediate

reward is given by

$$r_t = -x_t^2 - a_t^2. \quad (7)$$

The goal is to maximize the total discounted reward, defined by Equation 1 or 2 for all x . Because the task is a linear quadratic regulator (LQR) problem, it is possible to calculate the optimal control rule. From the discrete-time Riccati equation, the optimum regulator is given by

$$a_t = -k_1 x_t, \text{ where } k_1 = 1 - \frac{2}{1 + 2\gamma + \sqrt{4\gamma^2 + 1}}. \quad (8)$$

The optimum value function is given by $V^*(x_t) = -k_2 x_t^2$, where k_2 is a some positive constant. In this experiment, the set of possible states is constrained to lie in the range $[-4, 4]$. When the state transition given by Equation 6 does not result in the range $[-4, 4]$, the x_t is truncated. When the agent chooses an action that is not lie in the range $[-4, 4]$, the action executed in the environment is also truncated.

5.2 Implementation for the LQR Problem

5.2.1 The Actor

Remember the policy $\pi(a, W, X)$ is a probability density function when the set of possible action is continuous. The normal distribution is a simple multiparameter distribution for a continuous random variable. It has two parameters, the mean μ and the standard deviation σ . When the policy function π is given by the equation 9, the eligibility of μ and σ are

$$\pi(a, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a - \mu)^2}{2\sigma^2}\right) \quad (9)$$

$$e_\mu = \frac{a_t - \mu}{\sigma^2} \quad (10)$$

$$e_\sigma = \frac{(a_t - \mu)^2 - \sigma^2}{\sigma^3}. \quad (11)$$

One useful feature of such a *Gaussian unit* [Williams 92] is that the agent has a potential to control its degree of exploratory behavior. We must draw attention to the fact that the eligibility is to divergent when σ goes close to 0, because the parameter σ is occupying the denominators of Equation 10 and 11. The divergence of the eligibility has a bad influence on the algorithm. One way to overcome this problem is to control the step size of the update parameter vector using σ . It is obtained by setting the learning rate parameter proportional to σ^2 , then the eligibility can be seen as

$$e_\mu = a_t - \mu, e_\sigma = \frac{(a_t - \mu)^2 - \sigma^2}{\sigma}. \quad (12)$$

The actor would first compute μ and σ deterministically and then draw its output from the normal distribution that follows mean equal to μ and standard

deviation equal to σ . The actor has two internal variables, w_1 and w_2 , and computes the values of μ and σ according to

$$\mu = w_1 x_t, \quad \sigma = \frac{1}{1 + \exp(-w_2)}. \quad (13)$$

Then, w_1 can be seen as a feedback gain. The reason for this calculation of σ is to guarantee the σ to keep positive. The e_1 and e_2 are the characteristic eligibilities of w_1 and w_2 respectively. From Equation 12, e_1 and e_2 are given by

$$e_1 = e_\mu \frac{\partial}{\partial w_1} \mu = (a_t - \mu) x_t, \quad (14)$$

$$e_2 = e_\sigma \frac{\partial}{\partial w_2} \sigma = ((a_t - \mu)^2 - \sigma^2)(1 - \sigma). \quad (15)$$

The w_1 is initialized to 0.35 ± 0.15 , and $w_2 = 0$, i.e., $\sigma = 0.5$. The learning rate α_p is fixed to 0.001.

5.2.2 The Critic

The critic quantizes the continuous state-space ($-4 \leq x \leq 4$) into an array of boxes. We have tried two types of the quantizing: one is discretizing x evenly into 3 boxes, the other is 10 boxes. And the critic attempts to store in each box a prediction of the value \hat{V} by using TD(0) [Sutton 88]. The learning rate α for TD(0) is fixed to 0.2.

5.3 Simulation Results

Figure 4, 5, 6, 7 and 8 show the performance of 100 trials in the LQR problem with the discount rate $\gamma = 0.9$.

Figure 4 shows the performance of the algorithm, in which the critic uses 3 boxes, the actor does not use eligibility traces, i.e., $\beta = 0$. Figure 6 shows the performance where the critic uses 10 boxes, the actor does not use the traces. The algorithm in Figure 6 converged close to the optimum feedback gain. In contrast, Figure 4 didn't. The reason for this is that the ability of the function approximation (3 boxes) is insufficient for learning policy without the trace.

Figure 5 shows the performance where the critic uses 3 boxes, the actor uses the trace, $\beta = \gamma = 0.9$. It achieved much better results in terms of both the learning efficiency and the quality of the mean value of the converged policy than the algorithm in Figure 4 or 5. Obviously, the actor's eligibility trace relates these two advantages. The reason for the learning efficiency in this case may be that the actor's trace accelerates propagating information. The better quality of the policy is clearly owing to the property that the actor improves its policy by using a gradient of actual return, shown in Section 4.3. Therefore, the algorithm

using the trace was not influenced by the critic's ability in terms of the quality of the mean of the policy. We can also see this property in Figure 8, but its deviation is considerably large. Figure 9 shows the value function that is defined by Equation 1 and 7 over the parameter space μ and σ . The value of performance is fairly flat around the optimal solution. This is the reason that the deviation of the policy is large in Figure 8. This example makes it clear that the critic controls step-size of the actor's backups so that the step-size is taken to be smaller around the local maximum.

The algorithm in Figure 7 achieved best results in terms of both the mean and the deviation of the policy. The reason for this may be owing to the critic's perfect value estimation.

In this preliminary experiment, we can see that the algorithm using the actor's eligibility trace performed better than the algorithm without using the trace in the same computational resources.

Here we presented the results of the actor-critic that use only TD(0) in the critic, but we have also experimented on TD(λ) where $0 < \lambda \leq 1$. Roughly speaking, we have poor performance when the λ approaches close to 1. It follows from this that the eligibility trace in the critic cannot make up for the critic's poor ability of function approximation. The details of the experiments using TD(λ) will appear in other papers.

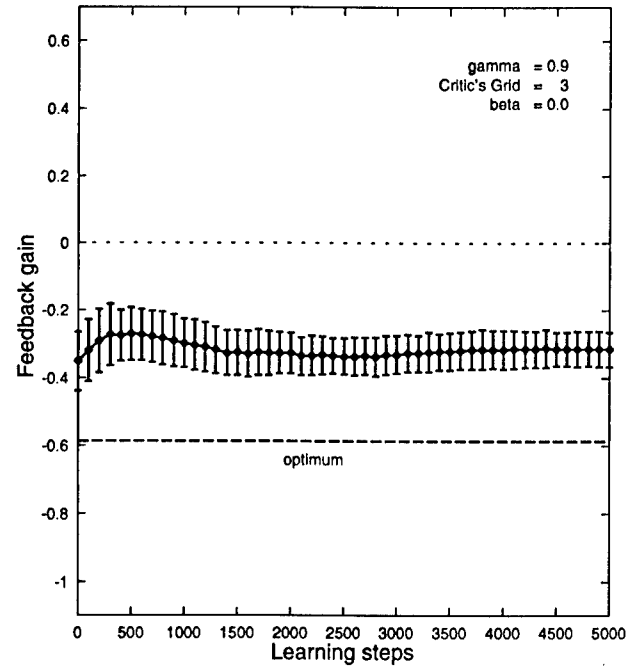


Figure 4: The average performance of 100 trials without the actor's eligibility trace ($\beta = 0$). The critic uses 3 boxes.

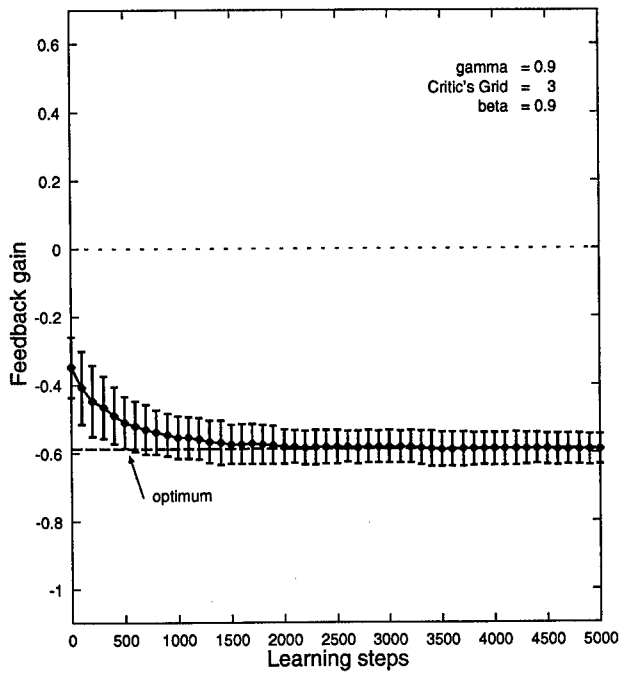


Figure 5: The average performance of 100 trials using the actor's trace $\beta = 0.9$. The critic uses 3 boxes.

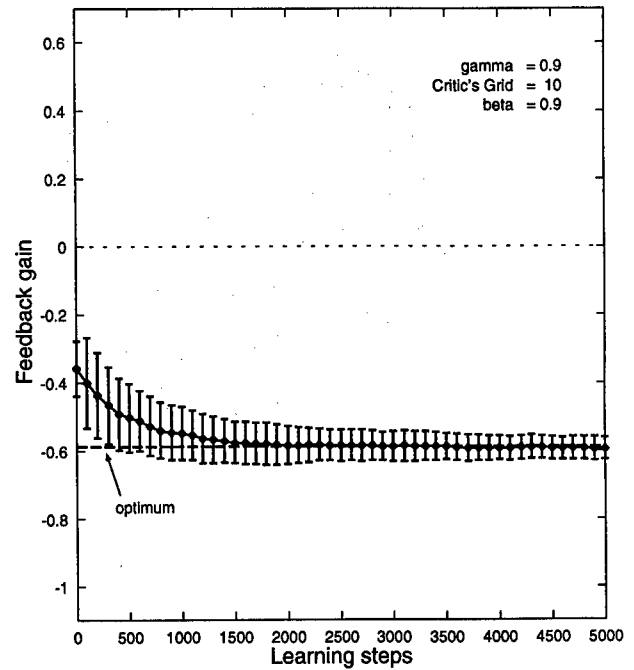


Figure 7: The average performance of 100 trials using the actor's trace $\beta = 0.9$. The critic uses 10 boxes.

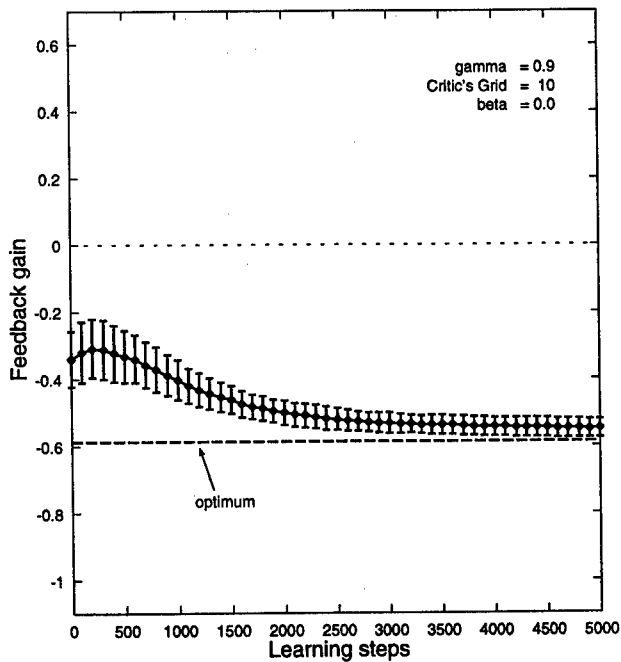


Figure 6: The average performance of 100 trials without the actor's trace ($\beta = 0$). The critic uses 10 boxes.

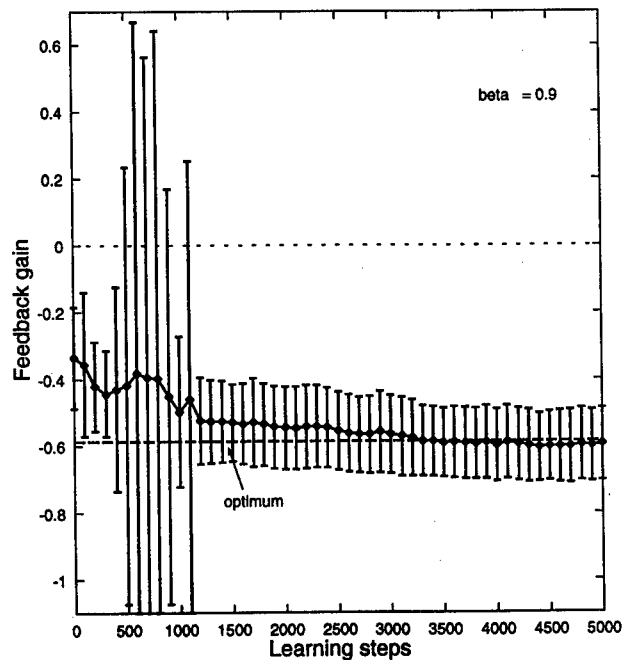


Figure 8: The average performance of 100 trials. $\beta = 0.9$. The agent learns without the critic, i.e., the critic provides $\hat{V}(x) = 0$ for all x .

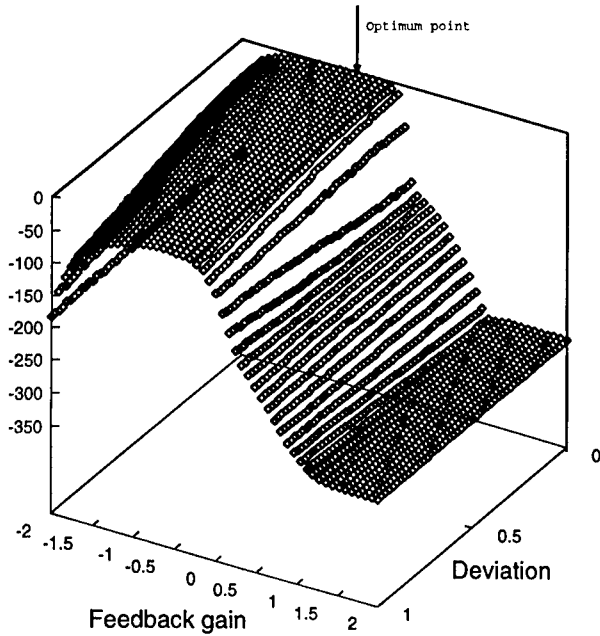


Figure 9: Value function over the parameter space in the LQR problem, where $\gamma = 0.9$. It is fairly flat around the optimum: $\mu = -0.5884$, $\sigma = 0$.

6 Applying to a Cart-Pole Problem

The behavior of this algorithm is demonstrated through a computer simulation of a cart-pole control task, that is a multi-dimensional nonlinear non-quadratic problem. We modified the cart-pole problem described in [Barto et al. 83] so that the action is taken to be continuous.

6.1 Problem Formulation

The dynamics of the cart-pole system is modeled by

$$\ddot{\theta} = \frac{g \sin \theta + \cos \theta \left(\frac{-F - m\ell \ddot{x} \sin \theta + \mu_c sgn(\dot{x})}{M+m} \right) - \frac{\mu_p \dot{\theta}}{m\ell}}{\ell \left(\frac{4}{3} - \frac{m \cos^2 \theta}{M+m} \right)},$$

$$\ddot{x} = \frac{F + m\ell (\ddot{\theta} \sin \theta - \dot{\theta}^2 \cos \theta) - \mu_c sgn(\dot{x})}{M+m},$$

where $M = 1.0$ (kg) denotes mass of the cart, $m = 0.1$ (kg) is mass of the pole, $2\ell = 1$ (m) is a length of the pole, $g = 9.8$ (m/sec^2) is the acceleration of gravity, F (N) denotes the force applied to cart's center of mass, $\mu_c = 0.0005$ is a coefficient of friction of cart, $\mu_p = 0.000002$ is a coefficient of friction of pole. In this simulation, we use discrete-time system to approximate these equations, where $\Delta t = 0.02$ sec. At each discrete time step, the agent observes $(x, \dot{x}, \theta, \dot{\theta})$, and controls the force F . The agent can execute action in

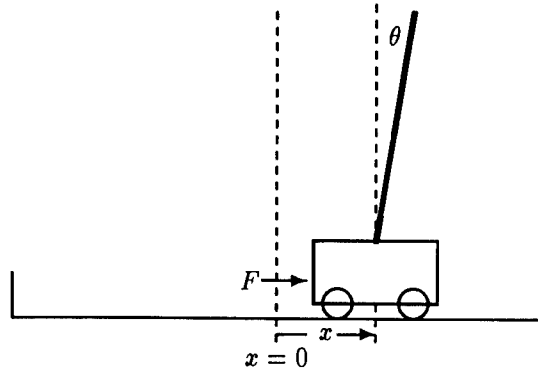


Figure 10: The cart-pole problem.

arbitrary range, but the possible action in the cart-pole system is constrained to lie in the range $[-20, 20](N)$. When the agent chooses an action which is not lie in that range, the action executed in the system is truncated. The system begins with $(x, \dot{x}, \theta, \dot{\theta}) = (0, 0, 0, 0)$. The system fails and receives a reward (penalty) signal of -1 when the pole falls over ± 12 degrees or the cart runs over the bounds of its track ($-2.4 \leq x \leq 2.4$), then the cart-pole system is reset to the initial state.

6.2 Details of the Agent

In this experiment, the actor adopts similar implementation shown in Equation 9 and 12. The state space is constrained in the range $(x, \dot{x}, \theta, \dot{\theta}) = (\pm 2.4 \text{ m}, \pm 2 \text{ m/sec}, \pm \pi \times 12/180 \text{ rad}, \pm 1.5 \text{ rad/sec})$. The actor has five internal variables $w_1 \dots w_5$, and computes the μ and σ according to

$$\mu = w_1 \frac{x_t}{2.4} + w_2 \frac{\dot{x}_t}{2} + w_3 \frac{\theta_t}{12\pi/180} + w_4 \frac{\dot{\theta}_t}{1.5},$$

$$\sigma = 0.1 + \frac{1}{1 + \exp(-w_5)}. \quad (16)$$

Similarly to Equation 14 and 15, the eligibilities $e_1 \dots e_5$ are given by

$$e_1 = (a_t - \mu) x_t, \quad e_2 = (a_t - \mu) \dot{x}_t$$

$$e_3 = (a_t - \mu) \theta_t, \quad e_4 = (a_t - \mu) \dot{\theta}_t$$

$$e_5 = ((a_t - \mu)^2 - \sigma^2)(1 + 0.1 - \sigma).$$

The critic discretizes the normalized state space evenly into $3 \times 3 \times 3 \times 3 = 81$ boxes, and attempts to store in each box \hat{V} by using TD(0) algorithm [Sutton 88]. The parameters are set to $\gamma = 0.95$, $\alpha = 0.5$, $\alpha_p = 0.001$.

6.3 Simulation Results

Figure 11 shows the performance of three learning algorithms in which the policy representation is the

same. The actor/critic algorithm using the actor's trace achieved best results. In contrast, the algorithm without using the trace couldn't learn the control policy because of the poor ability of function approximation in the critic.

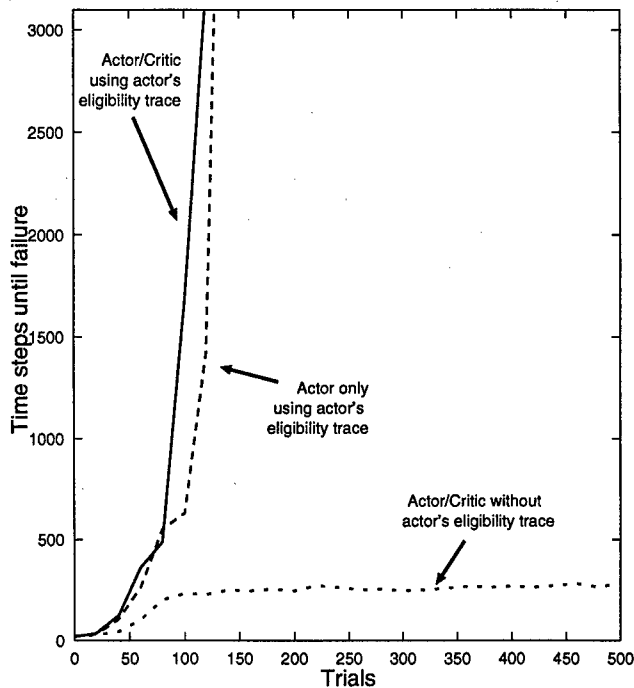


Figure 11: The average performance of three algorithms on 100 trials. The critic uses $3 \times 3 \times 3 \times 3$ boxes. A trial means an attempt from initial state to a failure.

7 Discussion

Representation of Policies: First of all, actor/critic algorithms should have sufficient ability to approximate policies. If it is satisfied, use of the actor's eligibility trace ($\beta = \gamma$) enables to learn an acceptable policy with less cost rather than increasing the critic's ability of function approximation in our test cases. The reason is that the policy function representation would require less memory than the representation of the state-action value function in many cases.

Controlling Step-Size of Backups: It is analytically shown in Section 4.3 that the critic provides an appropriate reinforcement baseline to the actor. The adaptive baseline controls step-size of the actor's backups so that the step-size is taken to be smaller around the local maximum. This property would contribute the better learning efficiency and the suppression of harmful drift of the policy that are shown in the experiments.

To Overcome non-Markovian: There are many ways to implement the critic's learning scheme. [Peng et al. 94] and [Sutton 95] pointed out that increasing λ makes $TD(\lambda)$ less sensitive to non-Markovian effect. The actor's eligibility traces are also useful in getting over non-Markovian problems [Kimura et al. 97]. Therefore, the combination of $TD(\lambda)$ and the actor's eligibility trace will be robust in non-Markovian problems.

Combining with Efficient DP-based Methods: If the hidden state is relatively small in the state space, the agent may perform good in which efficient DP-based algorithms are adopted for the critic. The DP-based algorithms accelerate the actor's learning in completely observable states, and the actor's stochastic policy and its trace ($\beta = \gamma$) would make up for the non-Markovian effects owing to the hidden state or function approximation.

8 Conclusions

This paper presented an analysis of actor/critic algorithms in which the actor updates its policy using the eligibility trace of the policy parameters. The results show that when the discount rate of the value function equals the discount factor of the actor's trace, the actor improves its policy by using a gradient of actual return, not by using a gradient of the estimated return in the critic. Then, the critic provides an adaptive reinforcement baseline to the actor controlling the step-size of the actor's backups. It enables the agent to learn a fairly good policy under the condition that the approximated value function in the critic is hopelessly imperfect. The behavior is demonstrated through simulations showing that the trace contributes the learning efficiency and the suppression of undesirable drifts of the policy. Analysis of the algorithm in non-Markovian environments is a future work.

Acknowledgements

We would like to thank Andrew Barto, Jing Peng, Jeff Schneider, Satinder Singh, Richard Sutton, and reviewers for many helpful comments and suggestions.

References

- [Baird 94] Baird, L. C.: Reinforcement Learning in Continuous Time: Advantage Updating, *Proceedings of IEEE International Conference on Neural Networks*, Vol. IV, pp. 2448-2453 (1994).
- [Barto et al. 83] Barto, A. G., Sutton, R. S. and Anderson, C. W.: Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no.5, September/October 1983, pp. 834-846.

- [Clouse et al. 92] Clouse, J. A. & Utogoff, P. E.: A Teaching Method for Reinforcement Learning, *Proc. of the 9th International Conference on Machine Learning*, pp. 93-101 (1992).
- [Crites et al. 94] Crites, R. H. and Barto, A. G.: An Actor/Critic Algorithm that is Equivalent to Q-Learning, *Advances in Neural Information Processing Systems 7*, pp. 401-408 (1994).
- [Doya 96] Doya, K.: Efficient Nonlinear Control with Actor-Tutor Architecture, *Advances in Neural Information Processing Systems 9*, pp. 1012-1018 (1996).
- [Gullapalli 92] Gullapalli, V.: Reinforcement Learning and Its Application to Control, *PhD Thesis*, University of Massachusetts, Amherst, COINS Technical Report 92-10 (1992).
- [Jaakkola 94] Jaakkola, T., Singh, S. P., & Jordan, M. I.: Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems, *Advances in Neural Information Processing Systems 7*, pp. 345-352 (1994).
- [Kaelbling et al. 96] Kaelbling, L. P., & Littman, M. L., & Moore, A. W.: Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237-277 (1996).
- [Kimura et al. 95] Kimura, H., Yamamura, M., & Kobayashi, S.: Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward, *Proceedings of the 12th International Conference on Machine Learning*, pp. 295-303 (1995).
- [Kimura et al. 97] Kimura, H., Miyazaki, K. and Kobayashi, S.: Reinforcement Learning in POMDPs with Function Approximation, *Proceedings of the 14th International Conference on Machine Learning*, pp. 152-160 (1997).
- [Lin et al. 96] Lin, C. J. and Lin, C. T.: Reinforcement Learning for An ART-Based Fuzzy Adaptive Learning Control Network, *IEEE Transactions on Neural Networks*, Vol. 7, No. 3, pp. 709-731 (1996).
- [Littman 94] Littman, M. L.: Markov games as a framework for multi-agent reinforcement learning, *Proc. of 11th International Conference on Machine Learning*, pp. 157-163 (1994).
- [Pendrith et al. 96] Pendrith, M. D. & Ryan, M. R. K.: Actual return reinforcement learning versus Temporal Differences: Some theoretical and experimental results, *Proceedings of the 13th International Conference on Machine Learning*, pp. 373-381 (1996).
- [Peng et al. 94] Peng, J. and Williams, R. J.: Incremental Multi-Step Q-Learning, *Proceedings of the 11th International Conference on Machine Learning*, pp. 226-232 (1994).
- [Singh 94] Singh, S. P., Jaakkola, T., & Jordan, M. I.: Learning Without State-Estimation in Partially Observable Markovian Decision Processes, *Proceedings of the 11th International Conference on Machine Learning*, pp. 284-292 (1994).
- [Singh 96] Singh, S. P., & Sutton, R. S.: Reinforcement Learning with Replacing Eligibility Traces, *Machine Learning 22*, pp. 123-158 (1996).
- [Sutton 88] Sutton, R. S.: Learning to Predict by the Methods of Temporal Differences, *Machine Learning 3*, pp. 9-44 (1988).
- [Sutton 90] Sutton, R. S.: Reinforcement Learning Architectures for Animats, *Proceedings of the 1st International Conference on Simulation of Adaptive Behavior*, pp. 288-295 (1990).
- [Sutton 95] Sutton, R. S.: TD Models: Modeling the world at a Mixture of Time Scales, *Proceedings of the 12th International Conference on Machine Learning*, pp. 531-539 (1995).
- [Sutton et al. 98] Sutton, R. S. & Barto, A.: Reinforcement Learning: An Introduction, *A Bradford Book*, The MIT Press (1998).
- [Watkins et al. 92] Watkins, C. J. C. H., & Dayan, P.: Technical Note: Q-Learning, *Machine Learning 8*, pp. 55-68 (1992).
- [Williams et al. 90] Williams, R. J. & Baird, L. C.: A Mathematical Analysis of Actor-Critic Architectures for Learning Optimal Controls through Incremental Dynamic Programming, *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pp. 96-101. Center for Systems Science, Dunham Laboratory, Yale University, New Haven (1990).
- [Williams 92] Williams, R. J.: Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning, *Machine Learning 8*, pp. 229-256 (1992).

Using learning for approximation in stochastic processes

Daphne Koller
 Computer Science Dept.
 Stanford University
 Stanford, CA 94305-9010
 koller@cs.stanford.edu

Raya Fratkina
 Computer Science Dept.
 Stanford University
 Stanford, CA 94305-9010
 raya@cs.stanford.edu

Abstract

To monitor or control a stochastic dynamic system, we need to reason about its current state. Exact inference for this task requires that we maintain a complete joint probability distribution over the possible states, an impossible requirement for most processes. Stochastic simulation algorithms provide an alternative solution by approximating the distribution at time t via a (relatively small) set of samples. The time t samples are used as the basis for generating the samples at time $t + 1$. However, since only existing samples are used as the basis for the next sampling phase, new parts of the space are never explored. We propose an approach whereby we try to generalize from the time t samples to unsampled regions of the state space. Thus, these samples are used as data for learning a distribution over the states at time t , which is then used to generate the time $t + 1$ samples. We examine different representations for a distribution, including density trees, Bayesian networks, and tree-structured Bayesian networks, and evaluate their appropriateness to the task. The machine learning perspective allows us to examine issues such as the tradeoffs of using more complex models, and to utilize important techniques such as regularization and priors. We validate the performance of our algorithm on both artificial and real domains, and show significant improvement in accuracy over the existing approach.

1 Introduction

In many real-world domains, we are interested in monitoring the evolution of a complex situation over time. For example, we may be monitoring a patient's vital signs in an ICU, analyzing a complex freeway traffic scene with the goal of controlling a moving vehicle, or even tracking motion of objects in a visual scene. Such systems have complex and unpredictable dynamics; thus, they are often modeled as stochastic dynamic systems. Even when a model of

the system is known, reasoning about the system is a computationally difficult task. Our main concern in this paper is in using machine learning techniques as part of a reasoning task; specifically, the task of *monitoring* the state of the system as it evolves and as new observations are obtained.

Theoretically, the monitoring task is straightforward. We simply maintain a probability distribution over the possible states at the current time. As time evolves, we update this distribution using the transition model; as new observations are obtained, we use Bayesian conditioning to update it. Such a distribution is called a *belief state*; in a Markovian process, it provides a concise summary of all of our past observations, and suffices both for predicting the future trajectory of the system as well as for making optimal decisions about our actions [Ast65].

Unfortunately, even systems whose evolution model is compactly represented rarely admit a compact representation of the belief state and an effective update process. Consider, for example, a stochastic system represented as a *dynamic Bayesian network (DBN)* [DK89]. A DBN partitions the evolution of the process into *time slices*, each of which represents a snapshot of the state of the system at one point in time. Like a Bayesian network (BN), the DBN utilizes a decomposed representation of the state via state variables and a graphical notation to represent the direct dependencies between the variables in the model. The evolution model of the system—the distribution over states at time $t + 1$ given the state at time t —is represented in a network fragment such as the one in Figure 1(a) (appropriately annotated with probabilities). DBNs have been used for a variety of applications, including freeway surveillance [FHKR95], monitoring complex factories [JKOP89], and more.

Exact inference algorithms for BNs have analogues for inference in DBNs [Kja92]. Unfortunately, in most cases, these algorithms also end up maintaining a belief state—a distribution over most or all of the variables in a time slice. Furthermore, it can be shown [BK98] that the belief state rarely has any structure that may support a compact representation. Thus, exact inference algorithms are forced to maintain a fully explicit joint distribution over an exponen-

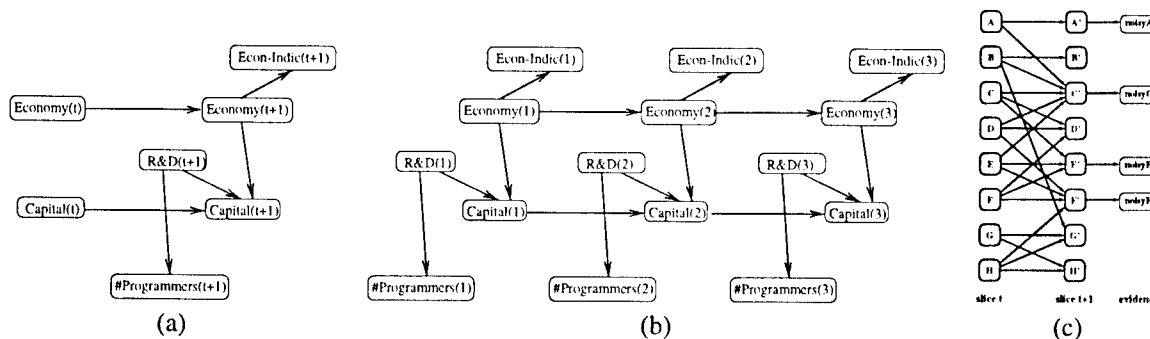


Figure 1: (a) The simple CAPITAL 2TBN for tracking the growth of a hi-tech company; (b) The same 2TBN unrolled for 3 time slices; (c) The WATER 2TBN.

tially large state space, making them impractical for most complex systems.

A similar problem arises when we attempt to monitor a process with complex continuous dynamics. Here also, an explicit representation of the belief state is infeasible.

This limitation has led to work on approximate inference algorithms for complex stochastic processes [GJ96, BK98, KKR95, IB96]. Of the approaches proposed, stochastic simulation algorithms are conceptually simplest and make the fewest assumptions about the structure of the process. The *survival of the fittest* (SOF) algorithm [KKR95] has been applied with success to large discrete DBNs [FHKR95]. The same algorithm (independently discovered by [IB96]) has been applied to the continuous problem of tracking object motion in cluttered visual scenes.

The algorithm, which builds on stochastic simulation algorithms for standard BNs [SP89], is as follows: For each time slice, we maintain a (small) set of weighted samples; a sample is one possible state of the system at that time, while its weight is some measure of how likely it is. This set of weighted samples is, in effect, a very sparse estimate of the belief state at time t . A sample at time t is propagated to time $t+1$ by a random process based on the dynamics of the system. In a naive generalization of [SP89], each time t sample is propagated forward to time $t+1$. However, as shown by [KKR95], this approach results in extremely poor performance, with the error of the approximation diverging rapidly as t grows. They propose an approach where samples are propagated preferentially: those whose weight is higher are more likely to be propagated, while the lower weight ones tend to be “killed off.” Technically, samples from time t are selected for propagation using a random process that chooses each sample proportionately to its weight. The resulting trajectories are weighted based on how well they fit the new evidence at time $t+1$, and the process continues. Despite its simplicity and low computational cost, the SOF algorithm performs very well; as shown in [KKR95], its error seems to remain bounded indefinitely over time. As shown in [IB96], this algorithm can also deal with complex continuous processes

much more successfully than standard techniques.

In this paper, we use machine learning techniques to improve the behavior of the SOF algorithm, with the goal of applying it to real-world complex domains. The SOF algorithm shifts its effort from less likely to more likely trajectories, thereby focusing on the more relevant parts of the space. However, at time $t+1$, it only samples parts of the space that arise from samples that it had at time t . Thus, it does not allow for correcting earlier mistakes: if its samples at time t were unrepresentative in some way, then its samples at time $t+1$ are also likely to be so. We can reinterpret this behavior from a somewhat different perspective. The set of time t samples are an approximation to the belief state at time t . When SOF chooses which samples to propagate it is simply sampling from this approximate belief state. Thus, the SOF algorithm is using a set of weighted samples as an approximation to a belief state, and a random sampling process to propagate an approximate belief state at time t to one at time $t+1$.

This perspective is a natural starting point for our approach. Clearly, a small number of weighted points is a suboptimal way of representing a complex distribution over a large space: As the number of samples is much smaller than the total size of the space, the representation is very sparse, and therefore necessarily unrepresentative. Our key insight is that our information about the relative likelihood of even a small number of points in the space can tell us a lot about the relative likelihood of others. Thus, we can treat our samples as data cases and use them to *learn* the shape of the distribution. In other words, we can use our own randomly generated samples as input to a density estimation algorithm, and use them to learn the distribution.

This insight leads us to explore a number of improvements to the SOF algorithm. We first note, in Section 3, that the number of samples needed to adequately estimate the distribution can vary widely: in situations where the evidence is unlikely, more samples will be needed in order to “find” relevant regions of the space. Luckily, as we are generating our own data, we can generate as many samples as we need; that is, we can perform a simple type of *active learning* [CAL94].

We then introduce a Dirichlet prior over the parameters of our distribution in order to deal with the problem of numerical overfitting, a particularly serious problem when we have a sparse sample for a very large space. We show that even these two simple improvements serve to significantly increase the accuracy of our algorithm.

We then proceed to investigate the issue of generalizing from the samples to other parts of the space. Of course, in order to generalize, we need a representation whose bias is higher. The requirements of our task impose several constraints both on the representation of the distribution and on the algorithm used to estimate it. First, as our state space is exponentially large, we must restrict attention to compact representations of distributions. Second, we must allow samples to be generated randomly from the distribution in a very efficient way. Thus, for example, a neural network whose input is a possible state of the process and whose output is the probability of that state would not be appropriate. Finally, as we are primarily interested in fast monitoring in time-critical applications, we prefer density estimation algorithms that are less compute-intensive.

Based on these constraints, we explore three main approaches, appropriate to processes represented as DBNs: Bayesian networks with a fixed structure, tree-structured Bayesian networks with a variable structure, and *density trees*, which resemble decision trees or discrete regression trees. We compare the performance of these algorithms to that of the SOF algorithm, and show that all three track the process with higher accuracy. We show that the density tree approach seems particularly promising, and suggest a possible explanation as to why it behaves better than the other approaches. We conclude with some discussion and possible extensions of our approach to other domains.

2 Preliminaries

A discrete time stochastic process is viewed as evolving randomly from one state to another at discrete time points. Formally, there is a set of states Σ such that at any point in time t , the situation can be described using some state $x \in \Sigma$. We typically assume that the process is *Markovian* so that the probability of being in state x' at time $t+1$ depends only on the state of the world at time t . Formally, letting $\mathbf{X}^{(t)}$ denote the random variable (or set of random variables) representing the state at time t , we have that $\mathbf{X}^{(t+1)}$ is independent of $\mathbf{X}^{(0)}, \dots, \mathbf{X}^{(t-1)}$ given $\mathbf{X}^{(t)}$. Thus, $P(\mathbf{X}^{(0)}, \dots, \mathbf{X}^{(t)}) = P(\mathbf{X}^{(0)}) \prod_{l=1}^t P(\mathbf{X}^{(l)} | \mathbf{X}^{(l-1)})$. We also typically assume that the process is time invariant, so that $P(\mathbf{X}^{(t)} | \mathbf{X}^{(t-1)})$ does not depend on t . Thus, it can be specified using a single *transition model* which holds for all time points.

In a DBN, the state of the process at time t is specified in terms of a set of state variables $X_1^{(t)}, \dots, X_n^{(t)}$. The transition model therefore has to define a probability distribution $P(X_1^{(t+1)}, \dots, X_n^{(t+1)} | X_1^{(t)}, \dots, X_n^{(t)})$. We specify such

a distribution using a network fragment called a *2TBN*—a two time-slice Bayesian network, as shown in Figure 1(a). A 2TBN defines the probability distribution for any time slice $t+1$ given time slice t : for each variable $X_i^{(t+1)}$ in the second time slice, the network fragment specifies a set of parents $\text{Parents}(X_i^{(t+1)})$, which can be variables either in time slice $t+1$ or in time slice t ; it also specifies a conditional probability table, which describes the probability distribution over the values of $X_i^{(t+1)}$ given any possible combination of values for its parents. As a whole, the 2TBN completely specifies $P(\mathbf{X}^{(t+1)} | \mathbf{X}^{(t)})$. The network fragment can be *unrolled* to define a distribution over arbitrarily many time slices. Figure 1(b) shows the 2TBN of Figure 1(a) unrolled over three time slices.

The state of the process is almost never fully observable; thus, in any time slice, we will get to observe the values of only some subset of the variables. In most monitoring tasks, the set of observable variables is the same in the different time slices. These variables typically represent sensor readings, e.g., the reading of some blood-pressure monitor in the ICU or the output of a video camera on a freeway overpass. Let $\mathcal{O}^{(t)}$ be the set of observable variables at time t , and let $\mathbf{o}^{(t)}$ be the instantiation of values for these variables observed at time t . In the *monitoring* task, we are interested in reasoning about $X_1^{(t)}, \dots, X_n^{(t)}$ given all the observations seen so far; i.e., we want to maintain $P(\mathbf{X}^{(t)} | \mathbf{o}^{(0)}, \dots, \mathbf{o}^{(t)})$.

As we discussed in the introduction, the stochastic simulation algorithms for DBNs is based on the standard likelihood weighting (LW) algorithm. The algorithm, shown in Figure 2, generates a sample by starting at the roots of the network and continuing in a top-down fashion, picking a value for every variable in turn. A value for a variable is sampled according to the appropriate conditional distribution, given the values already selected for its parents. Variables whose values were observed as evidence are not sampled; rather the variable is simply instantiated to its observed value. However, we must compensate for the fact that we forced this variable to take a value which may or may not be likely. Thus, we modify the weight of the sample to reflect the likelihood of having observed this particular value for this variable. It is easy to see that, while our algorithm only generates points x that are consistent with our observations, the expected weight for any such x (i.e., the probability with which it is generated times its weight when it is) is exactly its probability. Thus, our weighted samples are an unbiased estimator of the (unnormalized) distribution over the states and the observations. Note that the weight of the sample represents how well it explains our observations. Thus, a sample of very low weight is a very bad explanation for the observations, and contributes very little to our understanding of the situation.

The straightforward application of LW to DBNs is simply by treating the DBN as one very long BN. Roughly

```

LikelihoodWeighting( $\mathbf{x}^{(t)}, \mathbf{o}^{(t+1)}$ )
   $w := 1$ 
  for  $i := 1$  to  $n$ 
    Let  $\mathbf{u}$  be the assignment to  $\text{Parents}(X_i^{(t+1)})$  in  $\mathbf{x}^{(t)}, \mathbf{x}^{(t+1)}$ 
    If  $X_i^{(t+1)}$  is not in  $\mathbf{O}^{(t+1)}$ 
      Sample  $x_i^{(t+1)}$  from  $P(X_i^{(t+1)} \mid \text{Parents}(X_i^{(t+1)}) = \mathbf{u})$ 
    Else
      Set  $x_i^{(t+1)}$  to be  $X_i$ 's observed value in  $\mathbf{o}^{(t+1)}$ 
      Set  $w := w \cdot P(X_i^{(t+1)} = x_i^{(t+1)} \mid \text{Parents}(X_i^{(t+1)}) = \mathbf{u})$ 
  Return( $\mathbf{x}^{(t+1)}, w$ )

```

Figure 2: A temporal version of the likelihood weighting algorithm; it generates an instantiation $\mathbf{x}^{(t+1)}$ for the time $t + 1$ variables given an instantiation $\mathbf{x}^{(t)}$ for the time t variables.

speaking, the algorithm would maintain a set of samples $\mathbf{x}^{(t)}[1], \dots, \mathbf{x}^{(t)}[N]$ for every time slice t , representing possible states of the process at that time. At each time slice t , each of the samples is propagated to the next time slice using the LW algorithm, and its weight is adjusted according to how well it reflects the new observations. Unfortunately, as observed by [KKR95], this approach works very poorly for most DBNs. Intuitively, the process by which samples are randomly generated is oblivious to the evidence, which only affects the weight assigned to the samples. Therefore, the samples represent random trajectories through the system, most of which are completely irrelevant. As a consequence, as shown in [KKR95], the accuracy of LW diverges extremely quickly over time.

The *survival of the fittest* algorithm of [KKR95] addresses this problem by preferentially selecting which samples to propagate according to how likely they are, i.e., their weight relative to other samples. Technically, each sample $\mathbf{x}^{(t)}[j]$ is associated with a weight $w^{(t)}[j]$. In order to propagate to the next time slice, the algorithm first renormalizes all of the weights $w^{(t)}[j]$ to sum to 1. Then, it generates N new samples for time $t + 1$ as follows: For each new sample j , it selects $\mathbf{x}^{(t)}$ randomly from among $\mathbf{x}^{(t)}[1], \dots, \mathbf{x}^{(t)}[N]$, according to their weight. It then calls LW with $\mathbf{x}^{(t)}$ as a starting point, and gets back a time $t + 1$ sample $\mathbf{x}^{(t+1)}$ and a weight w . It sets $\mathbf{x}^{(t+1)}[j] := \mathbf{x}^{(t+1)}$ and $w^{(t+1)}[j] := w$. Note that the weight of the sample $w^{(t)}[j]$ manifests in the relative proportion with which $\mathbf{x}^{(t)}[j]$ will be propagated, so that we do not need to recount it when defining $w^{(t+1)}[j]$. Kanazawa *et al.* show empirically that, unlike LW, the error of SOF seems to remain bounded indefinitely over time.

3 Belief state estimation

We can interpret the SOF algorithm as estimating a probability distribution over the states at time t . Having generated some number of samples $\mathbf{x}^{(t)}[1], \dots, \mathbf{x}^{(t)}[N]$, it renormalizes their weights to sum to 1. The result is a simple

count distribution $\sigma_{sc}^{(t)}$ over the states at time t , one which gives some positive probability to states that correspond to one or more samples, and zero probability to all the rest. The SOF algorithm then generates N new samples from $\sigma_{sc}^{(t)}$, and propagates each of them to time $t + 1$ using the LW algorithm. The result samples are again renormalized, and the process repeats.

The distribution $\sigma_{sc}^{(t)}$ is a compact approximation to the belief state $\sigma^{(t)}$ at time t —the correct distribution $P(\mathbf{X}^{(t)} \mid \mathbf{o}^{(0)}, \dots, \mathbf{o}^{(t)})$. Assuming we know the initial state at time 0, $\sigma_{sc}^{(0)}$ is precisely the belief state at time 0. The properties of LW imply that our weighted samples at time 1 are an unbiased estimator for $P(\mathbf{X}^{(1)}, \mathbf{o}^{(1)} \mid \mathbf{o}^{(0)})$. Thus, after renormalization, $\sigma_{sc}^{(1)}$ is an estimator (albeit a biased one) for $P(\mathbf{X}^{(1)} \mid \mathbf{o}^{(0)}, \mathbf{o}^{(1)})$. By similar reasoning, we have that $\sigma_{sc}^{(t)}$ is a (biased) estimator for $P(\mathbf{X}^{(t)} \mid \mathbf{o}^{(0)}, \dots, \mathbf{o}^{(t)})$. However, each $\sigma_{sc}^{(t)}$ is only a very sparse approximation to $\sigma^{(t)}$, and thus one which is less than representative. It is also highly variable, with a strong dependence on which samples we happened to pick at the multiple previous random sampling stages. Both the sparsity and variability of our estimate propagate to the next time slice, increasing the variance of our approximation.

Our first attempt to control this variance relates to the amount of data on which our estimation is based. Naively, it seems that, at each phase, we are basing our estimation procedure on the same number N of samples. However, when we are renormalizing our distribution, we are not dividing by N , but rather by the total weight of the samples. Intuitively, if the evidence observed at a given time point is unlikely, each sample generated will explain it less well, so that its weight will be low. Thus, if the total weight of our N samples is low, then we have not really sampled a significant portion of the probability mass. Indeed, as argued by Dagum and Luby [DL97], the actual number of *effective* samples is their total weight. Thus, we modify the algorithm to guarantee that our estimation is based on a fixed weight rather than a fixed number of samples.

Our results for this improvement, applied to the simple CAPITAL network, are shown in Figure 3. The data were generated over 25 runs. In each run, the observations were generated randomly, from the correct distribution; thus they correspond to typical runs of the algorithm over a typical evidence sequence. Figure 3(a) shows the number of samples used over different time slices; we see that the number of samples varies significantly over time, illustrating that the algorithm is taking advantage of the additional flexibility. The average number of samples per time slice used over the run is 65. Figure 3(b) compares the accuracy of this algorithm to that of a fixed-samples algorithm using 65 samples in *each* time slice. We see that while the average number of samples used is the same, the variable-samples approach obtains consistently higher accuracy; in order to obtain comparable accuracy from the fixed-samples algo-

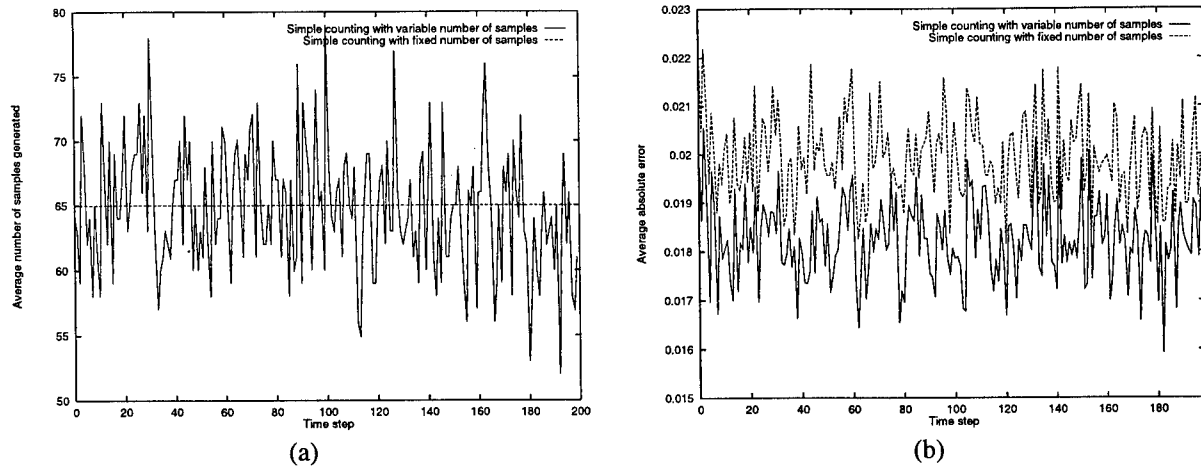


Figure 3: Comparison of variable-samples and fixed-samples algorithms for the CAPITAL network, averaged over 25 sequences: (a) number of samples used; (b) \mathcal{L}_1 error.

algorithm, around 70 samples are needed. We note that while both the error and the number of samples varies widely, they remain bounded indefinitely over time. This boundedness property continues to hold even in long runs with thousands of time slices. We also note that the number of states in the explicit belief state representation is 256, as compared to 55–80 samples used; thus our sampling approach allows considerable savings.

We have experimented with the number of samples required for different evidence sequences. Our results show that unlikely evidence sequences require many more samples than likely evidence, thereby justifying our intuition about the reason for the variability in the number of samples needed. Furthermore, the accuracy maintained by the variable-samples algorithm for likely and unlikely runs is essentially the same; thus, in a way, the algorithm generates as many samples as it needs to maintain a certain level of performance. We can view this ability as a type of *active learning* [CAL94], where the learning algorithm has the ability to ask for more data cases when necessary. In our context, the active learning paradigm is particularly appropriate, as the algorithm is generating its own data cases.

Our next improvement relates to another problem with the SOF algorithm. Our time t samples are necessarily very sparse, so that many entries in the probability distribution $\sigma_{sc}^{(t)}$ will have zero probability, even though their true probability is positive. This type of behavior can cause significant problems, as samples at time $t + 1$ are only generated based on our existing samples at time t . If the process is not very stochastic, i.e., if there are parts of the state space that only transition to other parts with very low probability, parts of the space that are not represented in $\sigma_{sc}^{(t)}$ will not be explored. Unfortunately, the parts of the space that are not represented may be quite likely; our sampling process may simply have missed them earlier, or they may be the results of trajectories that appeared unlikely in

earlier time slices because of misleading evidence. This problem is reflected clearly if we measure the distance between our approximation and the exact distribution using *relative entropy* [CT91], for many reasons the most appropriate distance measure for this type of situation. For an exact distribution ϕ and an approximate one ψ over the same space Ω , the relative entropy $D(\phi \parallel \psi)$ is defined as $\sum_{\omega \in \Omega} \phi(\omega) \log(\phi(\omega)/\psi(\omega))$. In cases, such as ours, where the approximate distribution ascribes probability 0 to entries that are not impossible, the relative entropy distance is infinite.

The machine learning perspective offers us a simple and theoretically well-founded solution to the problem of unwarranted zeros in our estimated distribution. We view the problem from the perspective of Bayesian learning, where we have a prior distribution over the parameters we are trying to estimate: the probabilities θ_x of the different states x in our belief state. An appropriate prior for multinomial distributions such as this is the *Dirichlet distribution*. We omit the formal definition of the Dirichlet prior, referring the reader to [Deg86]. Intuitively, it is defined using a set of hyperparameters α_{x_i} , each representing “imaginary” samples observed for the state x_i . In our case, as we have no beliefs in favor of one state x over another, we chose α_{x_i} to be uniformly α/r , where r is the total number of states consistent with our evidence.

Computing with this seemingly complex two-level distribution is actually quite simple, as most computations are equivalent to working with a single distribution $\sigma_{sc+}^{(t)}$, obtained from taking the *expectation* of the parameters $\{\theta_x\}$ relative to their prior distribution. In our case, for each x consistent with our evidence, we have that $\sigma_{sc+}^{(t)}(x) = (w^{(t)}(x) + \alpha/r)/Z$ where $w^{(t)}(x)$ is the total weight of samples $x^{(t)}[j]$ whose value is x , and Z is a normalizing factor. We see that each instantiation x in our distribution $\sigma_{sc+}^{(t)}$ (if consistent with our evidence) will have at

least some very small probability mass. We note that, even though $\sigma_{sc+}^{(t)}(x) > 0$ for every x , we need only represent explicitly those states which have materialized in our sampling algorithm. Thus, the cost of maintaining such a distribution is no higher than that of maintaining our original sparse set of samples.

The introduction of a prior serves to “spread out” some of the probability mass over unobserved states, increasing the amount of exploration done for unfamiliar regions of the space. We investigated the tradeoff between sampling in regions that are known to be likely and in new regions. As α grows, the performance of our algorithm first improves, then gradually decreases, as we would expect.

4 Alternative belief states representations

While this approach allows us to generate samples from unexplored parts of the space, it does so blindly: all unsampled states are treated in exactly the same way. However, our state space is not a completely arbitrary set of points. Two states x and x' which give the same values to almost all of the variables in our domain may be quite similar, and it may make sense to assume that their probabilities are much closer than that of other pairs. That is, we want to use our results for the states that we sampled to induce the probabilities of other states. This task is precisely a *density estimation* task (a type of unsupervised learning), where the set of sampled states are the training data.¹

As in any learning task, we must first define the hypothesis space. Essentially, our representations above fall into the category of *nonparametric* density estimators [Sco92]. (Roughly speaking, they are a discrete form of Parzen window.) As applied in our setting, these density estimators have no bias (and high variance); thus, they are incapable of generalizing from the training data to the rest of the space. In this section, we explore alternative representations of discrete densities that have higher bias and a correspondingly higher generalization power. As we discussed in the introduction, not every representation is suitable for our needs. Our representation must be significantly more compact than the full joint over the state variables; it must support an effective sampling process; and it must be easily learned. Two appropriate representations are Bayesian networks and *density trees*.

Bayesian networks. Given our overall problem, a Bayesian network representation for our distribution seems particularly appropriate. After all, our process is represented as a DBN, and is therefore highly structured. While it is known that conditional independences are not maintained in the belief states [BK98], it is reasonable to assume that some of the random variables in a time slice are

only weakly correlated with each other, and perhaps even weaker when conditioned on a third variable.

There has been a substantial amount of recent work on learning Bayesian networks from data (see [Hec95] for a survey). The simplest option is to fix the structure of the Bayesian network and to use our data to fill in the parameters for it. This process can be accomplished very efficiently, by a simple traversal over our data. Specifically, if our Bayesian network contains a node X with parents Y , then we need to estimate each of the parameters $P(X = x \mid Y = y)$. The *maximum likelihood* estimate for these parameters would be $\frac{\sigma_{sc+}(x, y)}{\sigma_{sc+}(y)}$. However, maximum likelihood estimates result in precisely the type of numerical overfitting (and particularly zero probability estimates) that we strove to avoid in the previous section. It turns out that, if we instead estimate the parameter as $\frac{\sigma_{sc+}(X=x, Y=y)}{\sigma_{sc+}(Y=y)}$, we get the effect of introducing a Dirichlet prior over each of our BN parameters. For a given Bayesian network structure B , the resulting distribution $\sigma_{bn(B)}$ is the one that minimizes $D(\sigma_{sc+} \parallel \sigma_{bn(B)})$ among all distributions representable by B .

One potential problem with this approach is that the BN structure B must be determined in advance, based on some prior knowledge of the user or on a manual analysis of the DBN structure. Furthermore the BN structure is fixed over the entire length of the run, whereas the true belief state $\sigma^{(t)}$ can change drastically as the process evolves. It seems quite likely that the most appropriate BN structure for approximating $\sigma^{(t)}$ also varies. This observation suggests that we select a different BN structure for each time slice. Unfortunately, learning of BN structure is a hard problem. Theoretically, even the problem of learning the optimal structure where each node is restricted to have at most k parents is NP-hard for any $k > 1$ [CHG95]. Pragmatically, the algorithms for this learning task are expensive, performing a greedy search, with multiple restarts, over the combinatorial (and superexponential) space of BN structures.

One option is to restrict our search to tree-structured BNs—ones where each node has at most one parent. Chow and Liu [CL68] present a simple (quadratic time) algorithm for finding the tree-structured BN whose distribution is closest—in terms of relative entropy—to the one in our data. The intuition is that, in a tree-structured BN, the edges should correspond to the strongest correlations. Thus, the algorithm introduces a direct connection between the variables whose *mutual information* [CT91] is largest. Formally, for each pair of variables X_i, X_j , we define an *edge-weight*

$$W(X_i, X_j) = \sum_{x_i, x_j} \sigma_{sc+}(x_i, x_j) \log \frac{\sigma_{sc+}(x_i, x_j)}{\sigma_{sc+}(x_i) \sigma_{sc+}(x_j)}$$

which is precisely the mutual information between X_i and X_j in σ_{sc+} . We then choose a maximum-weight spanning

¹If we view SOF as doing a process akin to bootstrapping by sampling from its own samples, our extension is akin to *smoothed bootstrapping* [Sil86].

tree over these nodes, where the weight of the tree is the sum of the weights of the edges it spans. We then select an arbitrary root for the tree, and fill in the conditional probability tables for the nodes in the network using σ_{sc+} (as in the case of a fixed BN). Here, again, choosing the parameters using σ_{sc+} is equivalent to introducing a Dirichlet prior over the network parameters. Chow and Liu show that the distribution σ_{st} represented by the resulting spanning tree minimizes $D(\sigma_{sc+} \parallel \sigma_{st})$.

Density trees. A *discrete density tree* is similar in overall structure to a classification decision tree. However, rather than representing a conditional distribution over some distinguished class variable given the features, the **density tree** represents a probability distribution σ_{dt} over some set of variables \mathbf{X} . Each interior node in the tree is labelled with a variable X , and the branches at that node with values x for that variable. A path on the tree to a node n thus corresponds to an assignment \mathbf{y}_n to some subset of the variables in the domain \mathbf{Y}_n .

The tree is a recursive representation of a multivariate distribution. At a high level, the tree structure partitions the space into bins, corresponding to the leaves in the tree. The distribution at each leaf n is uniform over the variables $\mathbf{X} - \mathbf{Y}_n$; the different leaf distributions are combined in a weighted average, where the weight of a leaf is simply the product of the edge-weights along the path to it. More technically, letting \mathbf{Z}_n represent $\mathbf{X} - \mathbf{Y}_n$, we have that if n is a leaf, then $\sigma_{dt}(\mathbf{Z}_n \mid n)$ is uniform over the values of \mathbf{Z}_n ; if n is an interior node labelled with X , then $\sigma_{dt}(\mathbf{Z}_n \mid n) = \sum_x \sigma_{dt}(X = x \mid n) \cdot \sigma_{dt}(\mathbf{Z}_n - \{X\} \mid n_x)$, where n_x is the child of n corresponding to the value x of X and $\sigma_{dt}(X = x \mid n)$ is the weight along the edge to it.

Our error function for the density tree learning algorithm is the relative entropy between our empirical distribution σ_{sc+} and σ_{dt} . We use a greedy algorithm which is very similar to the one for classification trees. We start out with the tree containing only the root node. We then iteratively split nodes on the variable that most decreases this error function. At each point, we estimate the parameters using σ_{sc+} , as we did for BNs. We use a greedy algorithm to determine the splits. The contribution that n makes to the overall relative entropy, if it remains a leaf, is proportional to $D(\sigma_{sc+}(\mathbf{Z}_n \mid \mathbf{y}_n) \parallel u_{\mathbf{Z}_n})$, where $u_{\mathbf{Z}_n}$ is the uniform distribution over the assignments \mathbf{z} to \mathbf{Z}_n . If we split n on a variable X , each of its children n_x (assuming they remain leaves) would make a contribution proportional to $\sigma_{sc+}(x \mid \mathbf{y}_n) D(\sigma_{sc+}(\mathbf{Z}_n - \{X\} \mid \mathbf{y}_n, x) \parallel u_{\mathbf{Z}_n - \{X\}})$. It is easy to show that the decrease in the relative entropy is precisely $D(\sigma_{sc+}(X \mid \mathbf{y}_n) \parallel u_X)$. We split n on that variable X which maximizes this decrease. Intuitively, this rule makes perfect sense: if we are representing the distributions at the leaves as uniform, then we should first extract these variables whose marginal distribution at n is the farthest from being uniform.

In order to avoid overfitting, we prevent the density-tree

	Relative error	#samples/slice	runtime/slice (minutes)
counting	$2.275 \pm 1.07 \times 10^{-4}$	1024 ± 1066	0.722
Chow-Liu tree	$2.106 \pm 0.75 \times 10^{-4}$	962 ± 977	0.044
BN 1 (29 params)	$2.102 \pm 0.96 \times 10^{-4}$	981 ± 970	0.064
BN 2 (340 params)	$2.104 \pm 0.79 \times 10^{-4}$	962 ± 1045	0.07
BN 3 (1401 params)	$2.112 \pm 0.73 \times 10^{-4}$	990 ± 1045	0.07
density tree	$1.816 \pm 0.89 \times 10^{-4}$	985 ± 1063	0.068

Figure 4: Means and standard deviations for different belief state representations

from growing to fit all of the samples. We utilize the standard idea of *early stopping*; our stopping rule prevents a node from splitting when the improvement to the relative entropy score is lower than some minimal amount. Specifically, we only allow a split of n on X when $\sigma_{sc+}(\mathbf{y}_n) \cdot D(\sigma_{sc+}(X \mid \mathbf{y}_n) \parallel u_X)$ is higher than some threshold.

We note that our notion of a density tree draws upon the literature of semiparametric density estimation techniques for continuous densities [Sco92]. The uniform distribution over samples at each leaf is similar to multi-dimensional histogram techniques; however, the tree structure allows variable-sized bins, and therefore greater flexibility in matching the number of parameters to the complexity of the distribution.

5 Experimental results

To provide a more realistic comparison, we tested the different variants of our algorithm on the practical WATER DBN [JKOP89], used for monitoring the biological processes of a water purification plant. (Comparable results were obtained for the CAPITAL network.) The WATER DBN had a substantially larger state space, with 27,648 possible values taken by the (non-evidence) variables. The structure of the WATER network is shown in Figure 1(c).

We experimented with several belief state representations: simple counting (SOF extended with priors); three different Bayesian networks of fixed structure, with 29, 340, and 1401 parameters respectively; Chow-Liu spanning trees; and density trees. We tested each representation on 10 runs, each of length 100, and where we used a variable-samples approach with a target weight of 5. For each run, we tested the average relative entropy error over the run.² (We also tested \mathcal{L}_1 error, with comparable results.) We then computed the mean and standard deviation of these run-average errors for the different representations. We did the same for the number of samples utilized per time slice. The results are shown in Figure 4.

Not surprisingly, the worst performer in terms of ac-

²We note that the momentary errors within a run—for belief states at individual time slices—can also vary widely, as can be seen from Figure 3. We tested the standard deviation of the momentary errors within a run, and it was approximately the same among all representations—around 50–55% of the overall average for the run. We omit the detailed results.

curacy is the simple counting approach. The performance of the Chow-Liu trees and the fixed Bayesian networks is about comparable, although the Bayesian network with a large number of parameters performs slightly worse than the rest. The density tree approach performs best, with a fairly significant margin. The number of samples used by the different approaches are not significantly different. What is significant is the fact that the number of samples generated is a factor of 15–25 smaller than the number of states in the state space. Indeed, the running times for the different approaches are all significantly lower than the 1.89 minutes per time slice required by exact inference. We note that the running times were all estimated on simple prototype code. We expect the running times for optimized code to be significantly lower. However, the relative efficiencies of the different algorithms should remain the same.

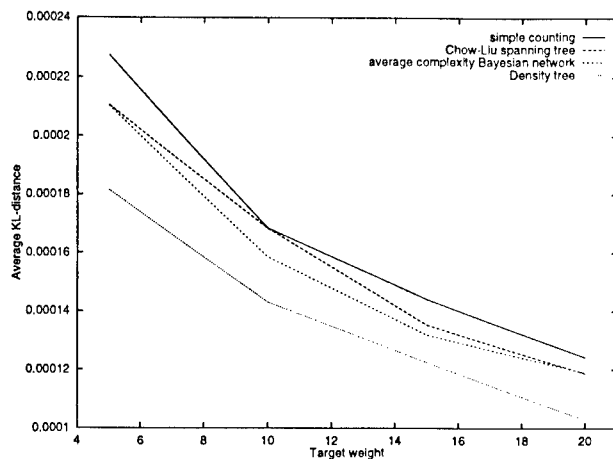


Figure 5: Average error for the WATER network for different target weights. The average is over 10 runs of 100 time slices each.

Figure 5 gives more evidence in favor of the density tree approach, demonstrating that it makes somewhat better use of data. The graph is a type of learning curve for the different approaches: the average error as a function of the target weight. We see that, for any given target weight, the density tree achieves higher accuracy. Furthermore, as we increase the target weight for our sampling algorithm, the error in the density tree approach decreases slightly faster. We note that this improvement does not come at the expense of increasing the overall number of samples: our experiments show that the average number of samples used is essentially identical for the different algorithms, and essentially linear in the target weight.

We believe that two factors contribute to making density trees a suitable representation for this task. The first is its inductive bias. A BN representation reflects an assumption that some of the random variables in the domain influence each other only weakly or indirectly via other variables. A density tree representation reflects an assumption that the distribution is substantially different when condi-

tioned on different values of the same variable. Our results indicate that the variability across different values of a variable is a more significant factor than any (weak) independences found in the distribution. We believe that the evidence serves to sharply skew the distribution in a certain direction, making it much more important for the approximate probability distribution to appropriately model that part of the space. Indeed, an examination of the trees produced by the density tree algorithm for different time slices shows that the parts of the tree corresponding to more likely parts of the space are usually represented using a much finer granularity—with subtrees that are two or more levels deeper—than the less likely ones.

A secondary factor that we believe also contributes to these performance results is the more flexible choice of the structure of the representation. This flexibility, which is shared by Chow-Liu trees and density trees, allows the representation of the approximate belief state to adapt to the current state of the process. An examination of the actual models learned by these algorithms at different points in time, shows that the structure does, in fact, vary significantly. This property is particularly helpful in the density tree case, as the most likely part of the state space changes in virtually every time slice.

6 Extensions and Conclusions

This paper deals with sampling-based approximate monitoring algorithms for a stochastic dynamic process. We have proposed the use of machine learning techniques in order to allow the algorithm to generalize from samples it has generated to samples it has not. We have shown that this idea can significantly improve the quality of our tracking for a given allocation of computational resources. We note that a related idea [BD97] has been proposed in the domain of combinatorial optimization algorithms, and has proved very effective. There, rather than maintaining a population of candidate solutions (as in genetic algorithms), the “good” candidate solutions generated by the algorithm are used to learn a distribution, from which samples are then generated for the next optimization phase.

We have investigated the use of several representations for our probability distributions. We saw that we get significant benefits from allowing the structure of the distribution to vary according to context—both for different parts of the space within the same distribution, and for different distributions over time. In our density tree representation, this flexibility was part of the definition. It would be interesting to see whether we could get even better performance by allowing the other representations to be more flexible. One possibility is to combine Bayesian networks and density trees; there are several ways of doing so, which we are currently investigating. We are also considering the use of other (computationally more expensive) representations of a density, e.g., as a mixture model where the mixture components have independent features [CS95].

It is interesting to also compare our approach to other types of algorithms for inference in stochastic processes. As we have shown, the number of samples generated by our algorithm is significantly lower than the number of states in the explicit representation of the belief state. Thus, our algorithm allows us to deal with domains in which exact inference is intractable. Another option is to use non-stochastic approximate inference algorithms [GJ96, BK98]. The approach of [GJ96] is not really intended for real-time monitoring, and is probably too computationally expensive to be used in that role. It also applies only to a fairly narrow class of stochastic models. The algorithm of [BK98] is more comparable to ours; essentially, it avoids the sampling step, directly propagating a time t approximate belief state to a time $t + 1$ approximate belief state. For certain types of processes, this approach probably dominates ours, as it avoids the additional variance introduced by the sampling phase. However, it is not obvious how it can be implemented effectively for all belief state representations (e.g., for density trees). Furthermore, it does not apply to processes where the representation of the process itself does not admit exact inference (e.g., highly-connected DBN models or models involving continuous variables).

By contrast, we note that our ideas are not specific to DBNs. The only use we made of the DBN model is as a representation from which we can generate random samples. We believe that our ideas apply to a much wider range of processes. Indeed, Isard and Blake [IB96] have obtained impressive results by using a stochastic sampling algorithm identical to simple SOF for the task of monitoring object motion in cluttered scenes. Here, the process is described using fairly complex continuous dynamics, that do not permit any exact inference algorithm. We believe that our ideas can also be used to provide improved algorithms for complex processes such as these, as well as for processes involving both continuous and discrete variables.

Acknowledgements

We would like to thank Eric Bauer and Xavier Boyen for allowing us to build on their code in our experiments. We would also like to thank Wray Buntine and Nir Friedman for useful comments and references. This research was supported by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant number DAAH04-96-1-0341, by ONR grant N00014-96-1-0718, and through the generosity of the Powell Foundation and the Sloan Foundation.

References

- [Ast65] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10:174–205, 1965.
- [BD97] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. ICML*, 1997.
- [BK98] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI*, 1998. To appear.
- [CAL94] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.
- [CHG95] D.M. Chickering, D. Heckerman, and D. Geiger. Learning Bayesian networks: Search methods and experimental results. In *Proc. AI & Stats*, pages 112–128, 1995.
- [CL68] C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transaction on Information Theory*, IT-14:462–467, 1968.
- [CS95] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. Fayyad et al., editor, *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1995.
- [CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.
- [Deg86] M.H. Degroot. *Probability and statistics*. Addison-Wesley, 1986.
- [DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Int.*, 5(3), 1989.
- [DL97] P. Dagum and M. Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93(1–2):1–27, 1997.
- [FHKR95] J. Forbes, T. Huang, K. Kanazawa, and S.J. Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proc. IJCAI*, pages 1878–1885, 1995.
- [GJ96] Z. Ghahramani and M.I. Jordan. Factorial hidden Markov models. In *NIPS* 8, 1996.
- [Hec95] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.
- [IB96] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. ECCV*, volume 1, pages 343–356, 1996.
- [JKOP89] F.V. Jensen, U. Kjærulff, K.G. Olesen, and J. Pedersen. An expert system for control of waste water treatment—a pilot project. Technical report, Judex Datasystemer A/S, Aalborg, Denmark, 1989.
- [Kja92] U. Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proc. UAI*, pages 121–129, 1992.
- [KKR95] K. Kanazawa, D. Koller, and S.J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proc. UAI*, pages 346–351, 1995.
- [Sco92] D.W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.
- [Sil86] B.W. Silverman. *Density estimation for statistics and data analysis*. Chapman & Hall, 1986.
- [SP89] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proc. UAI*, 1989.

An Information-Theoretic Definition of Similarity

Dekang Lin

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2

Abstract

Similarity is an important and widely used concept. Previous definitions of similarity are tied to a particular application or a form of knowledge representation. We present an information-theoretic definition of similarity that is applicable as long as there is a probabilistic model. We demonstrate how our definition can be used to measure the similarity in a number of different domains.

1 Introduction

Similarity is a fundamental and widely used concept. Many similarity measures have been proposed, such as information content [Resnik, 1995b], mutual information [Hindle, 1990], Dice coefficient [Frakes and Baeza-Yates, 1992], cosine coefficient [Frakes and Baeza-Yates, 1992], distance-based measurements [Lee et al., 1989; Rada et al., 1989], and feature contrast model [Tversky, 1977]. McGill *et al.* surveyed and compared 67 similarity measures used in information retrieval [McGill et al., 1979].

A problem with previous similarity measures is that each of them is tied to a particular application or assumes a particular domain model. For example, distance-based measures of concept similarity (e.g., [Lee et al., 1989; Rada et al., 1989]) assume that the domain is represented in a network. If a collection of documents is not represented as a network, the distance-based measures do not apply. The Dice and cosine coefficients are applicable only when the objects are represented as numerical feature vectors.

Another problem with the previous similarity measures is that their underlying assumptions are often not explicitly stated. Without knowing those assumptions, it is impossible to make theoretical arguments for or against any par-

ticular measure. Almost all of the comparisons and evaluations of previous similarity measures have been based on empirical results.

This paper presents a definition of similarity that achieves two goals:

Universality: We define similarity in information-theoretic terms. It is applicable as long as the domain has a probabilistic model. Since probability theory can be integrated with many kinds of knowledge representations, such as first order logic [Bacchus, 1988] and semantic networks [Pearl, 1988], our definition of similarity can be applied to many different domains where very different similarity measures had previously been proposed. Moreover, the universality of the definition also allows the measure to be used in domains where no similarity measure has previously been proposed, such as the similarity between ordinal values.

Theoretical Justification: The similarity measure is not defined directly by a formula. Rather, it is derived from a set of assumptions about similarity. In other words, if the assumptions are deemed reasonable, the similarity measure necessarily follows.

The remainder of this paper is organized as follows. The next section presents the derivation of a similarity measure from a set of assumptions about similarity. Sections 3 through 6 demonstrate the universality of our proposal by applying it to different domains. The properties of different similarity measures are compared in Section 7.

2 Definition of Similarity

Since our goal is to provide a formal definition of the intuitive concept of similarity, we first clarify our intuitions about similarity.

Intuition 1: The similarity between A and B is related to their commonality. The more commonality they share, the more similar they are.

Intuition 2: The similarity between A and B is related to the differences between them. The more differences they have, the less similar they are.

Intuition 3: The maximum similarity between A and B is reached when A and B are identical, no matter how much commonality they share.

Our goal is to arrive at a definition of similarity that captures the above intuitions. However, there are many alternative ways to define similarity that would be consistent with the intuitions. In this section, we first make a set of additional assumptions about similarity that we believe to be reasonable. A similarity measure can then be derived from those assumptions.

In order to capture the intuition that the similarity of two objects are related to their commonality, we need a measure of commonality. Our first assumption is:

Assumption 1: The commonality between A and B is measured by

$$I(\text{common}(A, B))$$

where $\text{common}(A, B)$ is a proposition that states the commonalities between A and B; $I(s)$ is the amount of information contained in a proposition s .

For example, if A is an orange and B is an apple. The proposition that states the commonality between A and B is "fruit(A) and fruit(B)". In information theory [Cover and Thomas, 1991], the information contained in a statement is measured by the negative logarithm of the probability of the statement. Therefore,

$$I(\text{common}(A, B)) = -\log P(\text{fruit}(A) \text{ and } \text{fruit}(B))$$

We also need a measure of the differences between two objects. Since knowing both the commonalities and the differences between A and B means knowing what A and B are, we assume:

Assumption 2: The differences between A and B is measured by

$$I(\text{description}(A, B)) - I(\text{common}(A, B))$$

where $\text{description}(A, B)$ is a proposition that describes what A and B are.

Intuition 1 and 2 state that the similarity between two objects are related to their commonalities and differences. We assume that commonalities and differences are the only factors.

Assumption 3: The similarity between A and B, $\text{sim}(A, B)$, is a function of their commonalities and dif-

ferences. That is,

$$\text{sim}(A, B) = f(I(\text{common}(A, B)), I(\text{description}(A, B)))$$

The domain of f is $\{(x, y) | x \geq 0, y > 0, y \geq x\}$.

Intuition 3 states that the similarity measure reaches a constant maximum when the two objects are identical. We assume the constant is 1.

Assumption 4: The similarity between a pair of identical objects is 1.

When A and B are identical, knowing their commonalities means knowing what they are, i.e., $I(\text{common}(A, B)) = I(\text{description}(A, B))$. Therefore, the function f must have the property: $\forall x > 0, f(x, x) = 1$.

When there is no commonality between A and B, we assume their similarity is 0, no matter how different they are. For example, the similarity between "depth-first search" and "leather sofa" is neither higher nor lower than the similarity between "rectangle" and "interest rate".

Assumption 5: $\forall y > 0, f(0, y) = 0$.

Suppose two objects A and B can be viewed from two independent perspectives. Their similarity can be computed separately from each perspective. For example, the similarity between two documents can be calculated by comparing the sets of words in the documents or by comparing their stylistic parameter values, such as average word length, average sentence length, average number of verbs per sentence, etc. We assume that the overall similarity of the two documents is a weighted average of their similarities computed from different perspectives. The weights are the amounts of information in the descriptions. In other words, we make the following assumption:

Assumption 6:

$$\forall x_1 \leq y_1, x_2 \leq y_2 : f(x_1 + x_2, y_1 + y_2) = \frac{y_1}{y_1 + y_2} f(x_1, y_1) + \frac{y_2}{y_1 + y_2} f(x_2, y_2)$$

From the above assumptions, we can prove the following theorem:

Similarity Theorem: The similarity between A and B is measured by the ratio between the amount of information needed to state the commonality of A and B and the information needed to fully describe what A and B are:

$$\text{sim}(A, B) = \frac{\log P(\text{common}(A, B))}{\log P(\text{description}(A, B))}$$

Proof:

$$\begin{aligned} & f(x, y) \\ &= f(x + 0, x + (y - x)) \\ &= \frac{x}{y} \times f(x, x) + \frac{y-x}{y} \times f(0, y-x) \quad (\text{Assumption 6}) \\ &= \frac{x}{y} \times 1 + \frac{y-x}{y} \times 0 = \frac{x}{y} \quad (\text{Assumption 4 and 5}) \end{aligned}$$

Q.E.D.

Since similarity is the ratio between the amount of information in the commonality and the amount of information in the description of the two objects, if we know the commonality of the two objects, their similarity tells us how much more information is needed to determine what these two objects are.

In the next 4 sections, we demonstrate how the above definition can be applied in different domains.

3 Similarity between Ordinal Values

Many features have ordinal values. For example, the “quality” attribute can take one of the following values “excellent”, “good”, “average”, “bad”, or “awful”. None of the previous definitions of similarity provides a measure for the similarity between two ordinal values. We now show how our definition can be applied here.

If “the quality of X is excellent” and “the quality of Y is average”, the maximally specific statement that can be said of both X and Y is that “the quality of X and Y are between “average” and “excellent”. Therefore, the commonality between two ordinal values is the interval delimited by them.

Suppose the distribution of the “quality” attribute is known (Figure 1). The following are four examples of similarity calculations:

$$\begin{aligned}
 \text{sim}(\text{excellent}, \text{good}) &= \frac{2 \times \log P(\text{excellent} \vee \text{good})}{\log P(\text{excellent}) + \log P(\text{good})} \\
 &= \frac{2 \times \log(0.05 + 0.10)}{\log 0.05 + \log 0.10} = 0.72 \\
 \text{sim}(\text{good}, \text{average}) &= \frac{2 \times \log P(\text{good} \vee \text{average})}{\log P(\text{average}) + \log P(\text{good})} \\
 &= \frac{2 \times \log(0.10 + 0.50)}{\log 0.10 + \log 0.50} = 0.34 \\
 \text{sim}(\text{excellent}, \text{average}) &= \frac{2 \times \log P(\text{excellent} \vee \text{good} \vee \text{average})}{\log P(\text{excellent}) + \log P(\text{average})} \\
 &= \frac{2 \times \log(0.05 + 0.10 + 0.50)}{\log 0.05 + \log 0.50} = 0.23 \\
 \text{sim}(\text{good}, \text{bad}) &= \frac{2 \times \log P(\text{good} \vee \text{average} \vee \text{bad})}{\log P(\text{good}) + \log P(\text{bad})} \\
 &= \frac{2 \times \log(0.10 + 0.50 + 0.20)}{\log 0.10 + \log 0.20} = 0.11
 \end{aligned}$$

The results show that, given the probability distribution in Figure 1, the similarity between “excellent” and “good” is much higher than the similarity between “good” and “average”; the similarity between “excellent” and “average” is much higher than the similarity between “good” and “bad”.

4 Feature Vectors

Feature vectors are one of the simplest and most commonly used forms of knowledge representation, especially in case-based reasoning [Aha et al., 1991; Stanfill and Waltz, 1986] and machine learning. Weights are often assigned to features to account for the fact that the dissimilarity caused by more important features is greater than the dissimilarity

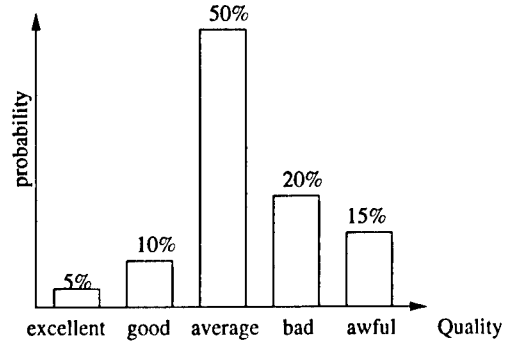


Figure 1: Example Distribution of Ordinal Values

caused by less important features. The assignment of the weight parameters is generally heuristic in nature in previous approaches. Our definition of similarity provides a more principled approach, as demonstrated in the following case study.

4.1 String Similarity—A case study

Consider the task of retrieving from a word list the words that are derived from the same root as a given word. For example, given the word “eloquently”, our objective is to retrieve the other related words such as “ineloquent”, “ineloquently”, “eloquent”, and “eloquence”. To do so, assuming that a morphological analyzer is not available, one can define a similarity measure between two strings and rank the words in the word list in descending order of their similarity to the given word. The similarity measure should be such that words derived from the same root as the given word should appear early in the ranking.

We experimented with three similarity measures. The first one is defined as follows:

$$\text{sim}_{\text{edit}}(x, y) = \frac{1}{1 + \text{editDist}(x, y)}$$

where $\text{editDist}(x, y)$ is the minimum number of character insertion and deletion operations needed to transform one string to the other.

The second similarity measure is based on the number of different trigrams in the two strings:

$$\text{sim}_{\text{tri}}(x, y) = \frac{1}{1 + |\text{tri}(x)| + |\text{tri}(y)| - 2 \times |\text{tri}(x) \cap \text{tri}(y)|}$$

where $\text{tri}(x)$ is the set of trigrams in x . For example, $\text{tri}(\text{eloquent}) = \{\text{elo}, \text{loq}, \text{oqu}, \text{que}, \text{ent}\}$.

Table 1: Top-10 Most Similar Words to "grandiloquent"

Rank	sim _{edit}		sim _{tri}		sim	
1	grandiloquently	1/3	grandiloquently	1/2	grandiloquently	0.92
2	grandiloquence	1/4	grandiloquence	1/4	grandiloquence	0.89
3	magniloquent	1/6	eloquent	1/8	eloquent	0.61
4	gradient	1/6	grand	1/9	magniloquent	0.59
5	grandaunt	1/7	grande	1/10	ineloquent	0.55
6	gradients	1/7	rand	1/10	eloquently	0.55
7	grandiose	1/7	magniloquent	1/10	ineloquently	0.50
8	diluent	1/7	ineloquent	1/10	magniloquence	0.50
9	ineloquent	1/8	grands	1/10	eloquence	0.50
10	grandson	1/8	eloquently	1/10	ventriloquy	0.42

Table 2: Evaluation of String Similarity Measures

Root	Meaning	W _{root}	11-point average precisions		
			sim _{edit}	sim _{tri}	sim
agog	leader, leading, bring	23	37%	40%	70%
cardi	heart	56	18%	21%	47%
circum	around, surrounding	58	24%	19%	68%
gress	to step, to walk, to go	84	22%	31%	52%
loqu	to speak	39	19%	20%	57%

The third similarity measure is based on our proposed definition of similarity under the assumption that the probability of a trigram occurring in a word is independent of other trigrams in the word:

$$\text{sim}(x, y) = \frac{2 \times \sum_{t \in \text{tri}(x) \cap \text{tri}(y)} \log P(t)}{\sum_{t \in \text{tri}(x)} \log P(t) + \sum_{t \in \text{tri}(y)} \log P(t)}$$

Table 1 shows the top 10 most similar words to "grandiloquent" according to the above three similarity measures.

To determine which similarity measure ranks higher the words that are derived from the same root as the given word, we adopted the evaluation metrics used in the Text Retrieval Conference [Harman, 1993]. We used a 109,582-word list from the AI Repository.¹ The probabilities of trigrams are estimated by their frequencies in the words. Let W denote the set of words in the word list and W_{root} denote the subset of W that are derived from root . Let (w_1, \dots, w_n) denote the ordering of $W - \{w\}$ in descending similarity to w according to a similarity measure. The precision of (w_1, \dots, w_n) at recall level $N\%$ is defined as the maximum value of $\frac{|W_{\text{root}} \cap \{w_1, \dots, w_k\}|}{k}$ such that $k \in \{1, \dots, n\}$ and $\frac{|W_{\text{root}} \cap \{w_1, \dots, w_k\}|}{|W_{\text{root}}|} \geq N\%$. The quality of the sequence (w_1, \dots, w_n) can be measured by the

11-point average of its precisions at recall levels 0%, 10%, 20%, ..., and 100%. The average precision values are then averaged over all the words in W_{root} . The results on 5 roots are shown in Table 2. It can be seen that much better results were achieved with sim than with the other similarity measures. The reason for this is that sim_{edit} and sim_{tri} treat all characters or trigrams equally, whereas sim is able to automatically take into account the varied importance in different trigrams.

5 Word Similarity

In this section, we show how to measure similarities between words according to their distribution in a text corpus [Pereira et al., 1993]. Similar to [Alshawhi and Carter, 1994; Grishman and Sterling, 1994; Ruge, 1992], we use a parser to extract dependency triples from the text corpus. A dependency triple consists of a head, a dependency type and a modifier. For example, the dependency triples in "I have a brown dog" consist of:

- (1) (have subj I), (have obj dog), (dog adj-mod brown), (dog det a)

where "subj" is the relationship between a verb and its subject; "obj" is the relationship between a verb and its object;

¹<http://www.cs.cmu.edu/afs/cs/project/ai-repository>

“adj-mod” is the relationship between a noun and its adjective modifier and “det” is the relationship between a noun and its determiner.

We can view dependency triples extracted from a corpus as features of the heads and modifiers in the triples. Suppose (avert obj duty) is a dependency triple, we say that “duty” has the feature obj-of(avert) and “avert” has the feature obj(duty). Other words that also possess the feature obj-of(avert) include “default”, “crisis”, “eye”, “panic”, “strike”, “war”, etc., which are also used as objects of “avert” in the corpus.

Table 3 shows a subset of the features of “duty” and “sanction”. Each row corresponds to a feature. A ‘x’ in the “duty” or “sanction” column means that the word possesses that feature.

Table 3: Features of “duty” and “sanction”

Feature	duty	sanction	$I(f_i)$
f_1 : subj-of(include)	x	x	3.15
f_2 : obj-of(assume)	x		5.43
f_3 : obj-of(avert)	x	x	5.88
f_4 : obj-of(ease)		x	4.99
f_5 : obj-of(impose)	x	x	4.97
f_6 : adj-mod(fiduciary)	x		7.76
f_7 : adj-mod(punitive)	x	x	7.10
f_8 : adj-mod(economic)		x	3.70

Let $F(w)$ be the set of features possessed by w . $F(w)$ can be viewed as a description of the word w . The commonalities between two words w_1 and w_2 is then $F(w_1) \cap F(w_2)$.

The similarity between two words is defined as follows:

$$(2) \text{sim}(w_1, w_2) = \frac{2 \times I(F(w_1) \cap F(w_2))}{I(F(w_1)) + I(F(w_2))}$$

where $I(S)$ is the amount of information contained in a set of features S . Assuming that features are independent of one another, $I(S) = -\sum_{f \in S} \log P(f)$, where $P(f)$ is the probability of feature f . When two words have identical sets of features, their similarity reaches the maximum value of 1. The minimum similarity 0 is reached when two words do not have any common feature.

The probability $P(f)$ can be estimated by the percentage of words that have feature f among the set of words that have the same part of speech. For example, there are 32868 unique nouns in a corpus, 1405 of which were used as subjects of “include”. The probability of subj-of(include) is $\frac{1405}{32868}$. The probability of the feature adj-mod(fiduciary) is $\frac{14}{32868}$ because only 14 (unique) nouns were modified by “fiduciary”. The amount of information in the feature adj-mod(fiduciary), 7.76, is greater than the amount of infor-

mation in subj-of(include), 3.15. This agrees with our intuition that saying that a word can be modified by “fiduciary” is more informative than saying that the word can be the subject of “include”.

The fourth column in Table 3 shows the amount of information contained in each feature. If the features in Table 3 were all the features of “duty” and “sanction”, the similarity between “duty” and “sanction” would be:

$$\frac{2 \times I(\{f_1, f_3, f_5, f_7\})}{I(\{f_1, f_2, f_3, f_5, f_6, f_7\}) + I(\{f_1, f_3, f_4, f_5, f_7, f_8\})}$$

which is equal to 0.66.

We parsed a 22-million-word corpus consisting of Wall Street Journal and San Jose Mercury with a principle-based broad-coverage parser, called PRINCIPAR [Lin, 1993; Lin, 1994]. Parsing took about 72 hours on a Pentium 200 with 80MB memory. From these parse trees we extracted about 14 million dependency triples. The frequency counts of the dependency triples are stored and indexed in a 62MB dependency database, which constitutes the set of feature descriptions of all the words in the corpus. Using this dependency database, we computed pairwise similarity between 5230 nouns that occurred at least 50 times in the corpus.

The words with similarity to “duty” greater than 0.04 are listed in (3) in descending order of their similarity.

- (3) responsibility, position, sanction, tariff, obligation, fee, post, job, role, tax, penalty, condition, function, assignment, power, expense, task, deadline, training, work, standard, ban, restriction, authority, commitment, award, liability, requirement, staff, membership, limit, pledge, right, chore, mission, care, title, capability, patrol, fine, faith, seat, levy, violation, load, salary, attitude, bonus, schedule, instruction, rank, purpose, personnel, worth, jurisdiction, presidency, exercise.

The following is the entry for “duty” in the Random House Thesaurus [Stein and Flexner, 1984].

- (4) **duty** *n.* 1. obligation, responsibility; onus; business, province; 2. function, task, assignment, charge. 3. tax, tariff, customs, excise, levy.

The shadowed words in (4) also appear in (3). It can be seen that our program captured all three senses of “duty” in [Stein and Flexner, 1984].

Two words are a pair of respective nearest neighbors (RNNs) if each is the other’s most similar word. Our program found 622 pairs of RNNs among the 5230 nouns that

Table 4: Respective Nearest Neighbors

Rank	RNN	Sim
1	earnings profit	0.50
11	revenue sale	0.39
21	acquisition merger	0.34
31	attorney lawyer	0.32
41	data information	0.30
51	amount number	0.27
61	downturn slump	0.26
71	there way	0.24
81	fear worry	0.23
91	jacket shirt	0.22
101	film movie	0.21
111	felony misdemeanor	0.21
121	importance significance	0.20
131	reaction response	0.19
141	heroin marijuana	0.19
151	championship tournament	0.18
161	consequence implication	0.18
171	rape robbery	0.17
181	dinner lunch	0.17
191	turmoil upheaval	0.17
201	biggest largest	0.17
211	blaze fire	0.16
221	captive westerner	0.16
231	imprisonment probation	0.16
241	apparel clothing	0.15
251	comment elaboration	0.15
261	disadvantage drawback	0.15
271	infringement negligence	0.15
281	angler fishermen	0.14
291	emission pollution	0.14
301	granite marble	0.14
311	gourmet vegetarian	0.14
321	publicist stockbroker	0.14
331	maternity outpatient	0.13
341	artillery warplanes	0.13
351	psychiatrist psychologist	0.13
361	blunder fiasco	0.13
371	door window	0.13
381	counseling therapy	0.12
391	austerity stimulus	0.12
401	ours yours	0.12
411	procurement zoning	0.12
421	neither none	0.12
431	briefcase wallet	0.11
441	audition rite	0.11
451	nylon silk	0.11
461	columnist commentator	0.11
471	avalanche raft	0.11
481	herb olive	0.11
491	distance length	0.10
501	interruption pause	0.10
511	ocean sea	0.10
521	flying watching	0.10
531	ladder spectrum	0.09
541	lotto poker	0.09
551	camping skiing	0.09
561	lip mouth	0.09
571	mounting reducing	0.09
581	pill tablet	0.08
591	choir troupe	0.08
601	conservatism nationalism	0.08
611	bone flesh	0.07
621	powder spray	0.06

occurred at least 50 times in the parsed corpus. Table 4 shows every 10th RNN.

Some of the pairs may look peculiar. Detailed examination actually reveals that they are quite reasonable. For example, the 221 ranked pair is "captive" and "westerner". It is very unlikely that any manually created thesaurus will list them as near-synonyms. We manually examined all 274 occurrences of "westerner" in the corpus and found that 55% of them refer to westerners in captivity. Some of the bad RNNs, such as (avalanche, raft), (audition, rite), were due to their relative low frequencies,² which make them susceptible to accidental commonalities, such as:

- (5) The {avalanche, raft} {drifted, hit}
 To {hold, attend} the {audition, rite}.
 An uninhibited {audition, rite}.

6 Semantic Similarity in a Taxonomy

Semantic similarity [Resnik, 1995b] refers to similarity between two concepts in a taxonomy such as the WordNet [Miller, 1990] or CYC upper ontology. The semantic similarity between two classes C and C' is not about the classes themselves. When we say "rivers and ditches are similar", we are not comparing the set of rivers with the set of ditches. Instead, we are comparing a generic river and a generic ditch. Therefore, we define $\text{sim}(C, C')$ to be the similarity between x and x' if all we know about x and x' is that $x \in C$ and $x' \in C'$.

The two statements " $x \in C$ " and " $x' \in C'$ " are independent (instead of being assumed to be independent) because the selection of a generic C is not related to the selection of a generic C' . The amount of information contained in " $x \in C$ and $x' \in C'$ " is

$$-\log P(C) - \log P(C')$$

where $P(C)$ and $P(C')$ are probabilities that a randomly selected object belongs to C and C' , respectively.

Assuming that the taxonomy is a tree, if $x_1 \in C$ and $x_2 \in C_2$, the commonality between x_1 and x_2 is $x_1 \in C_0 \wedge x_2 \in C_0$, where C_0 is the most specific class that subsumes both C_1 and C_2 . Therefore,

$$\text{sim}(x_1, x_2) = \frac{2 \times \log P(C_0)}{\log P(C_1) + \log P(C_2)}$$

For example, Figure 2 is a fragment of the WordNet. The number attached to each node C is $P(C)$. The similarity

²They all occurred 50–60 times in the parsed corpus.

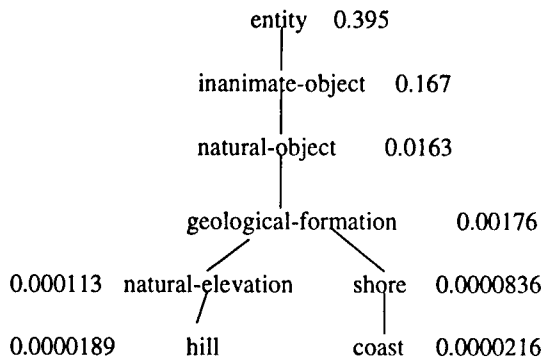


Figure 2: A Fragment of WordNet

between the concepts of Hill and Coast is:

$$\text{sim}(\text{Hill}, \text{Coast}) = \frac{2 \times \log P(\text{Geological-Formation})}{\log P(\text{Hill}) + \log P(\text{Coast})}$$

which is equal to 0.59.

There have been many proposals to use the distance between two concepts in a taxonomy as the basis for their similarity [Lee et al., 1989; Rada et al., 1989]. Resnik [Resnik, 1995b] showed that the distance-based similarity measures do not correlate to human judgments as well as his measure. Resnik's similarity measure is quite close to the one proposed here: $\text{sim}_{\text{Resnik}}(A, B) = \frac{1}{2} I(\text{common}(A, B))$. For example, in Figure 2, $\text{sim}_{\text{Resnik}}(\text{Hill}, \text{Coast}) = -\log P(\text{Geological-Formation})$.

Wu and Palmer [Wu and Palmer, 1994] proposed a measure for semantic similarity that could be regarded as a special case of $\text{sim}(A, B)$:

$$\text{sim}_{\text{Wu\&Palmer}}(A, B) = \frac{2 \times N_3}{N_1 + N_2 + 2 \times N_3}$$

where N_1 and N_2 are the number of IS-A links from A and B to their most specific common superclass C; N_3 is the number of IS-A links from C to the root of the taxonomy. For example, the most specific common superclass of Hill and Coast is Geological-Formation. Thus, $N_1 = 2$, $N_2 = 2$, $N_3 = 3$ and $\text{sim}_{\text{Wu\&Palmer}}(\text{Hill}, \text{Coast}) = 0.6$.

Interestingly, if $P(C|C')$ is the same for all pairs of concepts such that there is an IS-A link from C to C' in the taxonomy, $\text{sim}_{\text{Wu\&Palmer}}(A, B)$ coincides with $\text{sim}(A, B)$.

Resnik [Resnik, 1995a] evaluated three different similarity measures by correlating their similarity scores on 28 pairs of concepts in the WordNet with assessments made by human subjects [Miller and Charles, 1991]. We adopted

Table 5: Results of Comparison between Semantic Similarity Measures

Word Pair	Miller&Charles	Resnik	Wu & Palmer	sim
car, automobile	3.92	11.630	1.00	1.00
gem, jewel	3.84	15.634	1.00	1.00
journey, voyage	3.84	11.806	.91	.89
boy, lad	3.76	7.003	.90	.85
coast, shore	3.70	9.375	.90	.93
asylum, madhouse	3.61	13.517	.93	.97
magician, wizard	3.50	8.744	1.00	1.00
midday, noon	3.42	11.773	1.00	1.00
furnace, stove	3.11	2.246	.41	.18
food, fruit	3.08	1.703	.33	.24
bird, cock	3.05	8.202	.91	.83
bird, crane	2.97	8.202	.78	.67
tool, implement	2.95	6.136	.90	.80
brother, monk	2.82	1.722	.50	.16
crane, implement	1.68	3.263	.63	.39
lad, brother	1.66	1.722	.55	.20
journey, car	1.16	0	0	0
monk, oracle	1.10	1.722	.41	.14
food, rooster	0.89	.538	.7	.04
coast, hill	0.87	6.329	.63	.58
forest, graveyard	0.84	0	0	0
monk, slave	0.55	1.722	.55	.18
coast, forest	0.42	1.703	.33	.16
lad, wizard	0.42	1.722	.55	.20
chord, smile	0.13	2.947	.41	.20
glass, magician	0.11	.538	.11	.06
noon, string	0.08	0	0	0
rooster, voyage	0.08	0	0	0
Correlation with Miller & Charles	1.00	0.795	0.803	0.834

the same data set and evaluation methodology to compare $\text{sim}_{\text{Resnik}}$, $\text{sim}_{\text{Wu\&Palmer}}$ and sim . Table 5 shows the similarities between 28 pairs of concepts, using three different similarity measures. Column Miller&Charles lists the average similarity scores (on a scale of 0 to 4) assigned by human subjects in Miller&Charles's experiments [Miller and Charles, 1991]. Our definition of similarity yielded slightly higher correlation with human judgments than the other two measures.

7 Comparison between Different Similarity Measures

One of the most commonly used similarity measure is call Dice coefficient. Suppose two objects can be described with two numerical vectors (a_1, a_2, \dots, a_n) and

Table 6: Comparison between Similarity Measures

Property	Similarity Measures:				
	sim	WP	R	Dice	sim _{dist}
increase with commonality	yes	yes	yes	yes	no
decrease with difference	yes	yes	no	yes	yes
triangle inequality	no	no	no	no	yes
Assumption 6	yes	yes	no	yes	no
max value=1	yes	yes	no	yes	yes
semantic similarity	yes	yes	yes	no	yes
word similarity	yes	no	no	yes	yes
ordinal values	yes	no	no	no	no

(b_1, b_2, \dots, b_n) , their Dice coefficient is defined as

$$\text{sim}_{\text{dice}}(A, B) = \frac{2 \times \sum_{i=1, n} a_i b_i}{\sum_{i=1, n} a_i^2 + \sum_{i=1, n} b_i^2}.$$

Another class of similarity measures is based a distance metric. Suppose $\text{dist}(A, B)$ is a distance metric between two objects, sim_{dist} can be defined as follows:

$$\text{sim}_{\text{dist}}(A, B) = \frac{1}{1 + \text{dist}(A, B)}$$

Table 6 summarizes the comparison among 5 similarity measures.

Commonality and Difference: While most similarity measures increase with commonality and decrease with difference, sim_{dist} only decreases with difference and $\text{sim}_{\text{Resnik}}$ only takes commonality into account.

Triangle Inequality: A distance metrics must satisfy the triangle inequality:

$$\text{dist}(A, C) \leq \text{dist}(A, B) + \text{dist}(B, C).$$

Consequently, sim_{dist} has the property that $\text{sim}_{\text{dist}}(A, C)$ cannot be arbitrarily close to 0 if none of $\text{sim}_{\text{dist}}(A, B)$ and $\text{sim}_{\text{dist}}(B, C)$ is 0. This can be counter-intuitive in some situations. For example, in Figure 3, A and B are similar in

their shades, B and C are similar in their shape, but A and C are not similar.

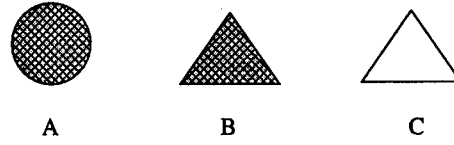


Figure 3: Counter-example of Triangle Inequality

Assumption 6: The strongest assumption that we made in Section 2 is Assumption 6. However, this assumption is not unique to our proposal. Both $\text{sim}_{\text{Wu\&Palmer}}$ and sim_{dice} also satisfy Assumption 6. Suppose two objects A and B are represented by two feature vectors (a_1, a_2, \dots, a_n) and (b_1, b_2, \dots, b_n) , respectively. Without loss of generality, suppose the first k features and the rest $n - k$ features represent two independent perspectives of the objects.

$$\begin{aligned} \text{sim}_{\text{dice}}(A, B) &= \frac{2 \times \sum_{i=1, n} a_i b_i}{\sum_{i=1, n} a_i^2 + \sum_{i=1, n} b_i^2} = \\ &= \frac{\sum_{i=1, k} a_i^2 + \sum_{i=1, k} b_i^2}{\sum_{i=1, n} a_i^2 + \sum_{i=1, n} b_i^2} \frac{2 \times \sum_{i=1, k} a_i b_i}{\sum_{i=1, k} a_i^2 + \sum_{i=1, k} b_i^2} + \\ &= \frac{\sum_{i=k+1, n} a_i^2 + \sum_{i=k+1, n} b_i^2}{\sum_{i=1, n} a_i^2 + \sum_{i=1, n} b_i^2} \frac{2 \times \sum_{i=k+1, n} a_i b_i}{\sum_{i=k+1, n} a_i^2 + \sum_{i=k+1, n} b_i^2} \end{aligned}$$

which is a weighted average of the similarity between A and B in each of the two perspectives.

Maximum Similarity Values: With most similarity measures, the maximum similarity is 1, except $\text{sim}_{\text{Resnik}}$, which have no upper bound for similarity values.

Application Domains: The similarity measure proposed in this paper can be applied in all the domains listed in Table 6, including the similarity of ordinal values, where none of the other similarity measures is applicable.

8 Conclusion

Similarity is an important and fundamental concept in AI and many other fields. Previous proposals for similarity measures are heuristic in nature and tied to a particular domain or form of knowledge representation. In this paper, we present a universal definition of similarity in terms of information theory. The similarity measure is not directly stated as in earlier definitions, rather, it is derived from a set of assumptions. In other words, if one accepts the assumptions, the similarity measure necessarily follows. The universality of the definition is demonstrated by its applications in different domains where different similarity measures have been employed before.

Acknowledgment

The author wishes to thank the anonymous reviewers for their valuable comments. This research has been partially supported by NSERC Research Grant OGP121338.

References

- [Aha et al., 1991] Aha, D., Kibler, D., and Albert, M. (1991). Instance-Based Learning Algorithms. *Machine Learning*, 6(1):37–66.
- [Alshawi and Carter, 1994] Alshawi, H. and Carter, D. (1994). Training and scaling preference functions for disambiguation. *Computational Linguistics*, 20(4):635–648.
- [Bacchus, 1988] Bacchus, F. (1988). *Representing and Reasoning with Probabilistic Knowledge*. PhD thesis, University of Alberta, Edmonton, Alberta, Canada.
- [Cover and Thomas, 1991] Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley series in telecommunications. Wiley, New York.
- [Frakes and Baeza-Yates, 1992] Frakes, W. B. and Baeza-Yates, R., editors (1992). *Information Retrieval, Data Structure and Algorithms*. Prentice Hall.
- [Grishman and Sterling, 1994] Grishman, R. and Sterling, J. (1994). Generalizing automatically generated selectional patterns. In *Proceedings of COLING-94*, pages 742–747, Kyoto, Japan.
- [Harman, 1993] Harman, D. (1993). Overview of the first text retrieval conference. In *Proceedings of SIGIR'93*, pages 191–202.
- [Hindle, 1990] Hindle, D. (1990). Noun classification from predicate-argument structures. In *Proceedings of ACL-90*, pages 268–275, Pittsburgh, Pennsylvania.
- [Lee et al., 1989] Lee, J. H., Kim, M. H., and Lee, Y. J. (1989). Information retrieval based on conceptual distance in is-a hierarchies. *Journal of Documentation*, 49(2):188–207.
- [Lin, 1993] Lin, D. (1993). Principle-based parsing without overgeneration. In *Proceedings of ACL-93*, pages 112–120, Columbus, Ohio.
- [Lin, 1994] Lin, D. (1994). Principar—an efficient, broad-coverage, principle-based parser. In *Proceedings of COLING-94*, pages 482–488. Kyoto, Japan.
- [McGill et al., 1979] McGill et al., M. (1979). An evaluation of factors affecting document ranking by information retrieval systems. Project report, Syracuse University School of Information Studies.
- [Miller, 1990] Miller, G. A. (1990). WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312.
- [Miller and Charles, 1991] Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.
- [Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, California.
- [Pereira et al., 1993] Pereira, F., Tishby, N., and Lee, L. (1993). Distributional Clustering of English Words. In *Proceedings of ACL-93*, pages 183–190, Ohio State University, Columbus, Ohio.
- [Rada et al., 1989] Rada, R., Mili, H., Bicknell, E., and Blettner, M. (1989). Development and application of a metric on semantic nets. *IEEE Transaction on Systems, Man, and Cybernetics*, 19(1):17–30.
- [Resnik, 1995a] Resnik, P. (1995a). Disambiguating noun groupings with respect to wordnet senses. In *Third Workshop on Very Large Corpora*. Association for Computational Linguistics.
- [Resnik, 1995b] Resnik, P. (1995b). Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of IJCAI-95*, pages 448–453, Montreal, Canada.
- [Ruge, 1992] Ruge, G. (1992). Experiments on linguistically based term associations. *Information Processing & Management*, 28(3):317–332.
- [Stanfill and Waltz, 1986] Stanfill, C. and Waltz, D. (1986). Toward Memory-based Reasoning. *Communications of ACM*, 29:1213–1228.
- [Stein and Flexner, 1984] Stein, J. and Flexner, S. B., editors (1984). *Random House College Thesaurus*. Random House, New York.
- [Tversky, 1977] Tversky, A. (1977). Features of similarity. *Psychological Review*, 84:327–352.
- [Wu and Palmer, 1994] Wu, Z. and Palmer, M. (1994). Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 133–138, Las Cruces, New Mexico.

Structural machine learning with Galois lattice and Graphs

Michel Liquiere

Lirmm,

161 rue ADA

34392 Montpellier cedex 5

France

Liquiere@lirmm.fr

Jean Sallantin

Lirmm,

161 rue ADA

34392 Montpellier cedex 5

France

Sallantin@lirmm.fr

Abstract

This paper defines a formal approach to learning from examples described by labelled graphs. We propose a formal model based upon lattice theory and in particular with the use of Galois lattice. We enlarge the domain of formal concept analysis, by the use of the Galois lattice model with structural description of examples and concepts. Our implementation, called "Gaal" (for GRAPh And Learning) constructs a Galois lattice for any description language provided that the two operations of comparison and generalization are determined for that language. We prove that these operations exist in the case of labelled graphs.

1. INTRODUCTION

The Galois lattice is the foundation of a set of conceptual classification methods. This approach defined by Barbut and Monjardet (Barbut, 1970), was popularized by (Wille, 1982), (Wille, 1992), who used this structure as the kernel of formal concept analysis.

Wille proposed considering each node of a Galois lattice as a formal concept. Each node has two parts: the extension (a subset of the examples) and the intension (the description). In addition, the lattice gives the relations (generalisation/specialization) between concepts. An advantages of this formalization is a good description of the concept space. Additionally, there are many methods for the construction of such lattice (depth first search (Bordat, 1986), incremental construction (Ganter, 1988), (Missikoff, 1989)).

In the context of machine learning, the automatic construction of such a hierarchy can be viewed as an unsupervised conceptual classification method as seen in (Michalski, 1982) because we give a general method and look for all the concepts that can be extracted from the examples.

In this way, research space is not limited by the use of parameters although this method cannot be used in all practical applications. Its advantage is that we can study precisely the impact of bias and heuristic.

An important limitation of the method using the Galois lattice is the classical propositional description of the examples (Wille, 1982), (Ganascia, 1993), (Mephu Nguifo, 1994), (Carpineto, 1994). There is a great deal of research on the extension of the description language: valued attributes (Wille, 1989), (Carpineto, 1994), term (Daniel-Vatonne, 1993), graph (Liquiere, 1989), (Godin, 1995).

In the case of structural description, the actual methods use a two step mechanism.

1) the goal of the first step is to find structures repeated in the set of descriptions of the examples.

2) the second step uses the structures found and changes the description of the example. Each structural description is converted in a list of binary attributes (one attribute by structure). An attribute is true if the associated structure appears in the example.

- in our work (Liquiere, 1989), (Liquiere, 1994), we used labelled graphs, and the goal of our first step was to find repeated paths and trees in the description of the examples.

- in the work of (Daniel-Vatonne, 1993), the description language is based upon rooted tree (term) and the first step research path.

- Godin, Mineau (Godin, 1995) uses a similar method with conceptual graphs.

The first step research repeated triplet graphs (graph like <Object>-relation-<Object>). This limits the complexity of the research. The second step finds sets of triplet graphs viewed in the same set of examples, but the link between the nodes are overlooked. So the structural descriptions of the examples are not exploited.

In this paper we give a general one step mechanism, without changing the description of the examples. This mechanism uses a generalization operation and we specify this operation for different classes of description languages.

This paper is organized as follows.

A generalized method of learning from examples is presented in section 2.

In section 3, we specify this model for description with labelled graphs.

Then, we study the complexity of the operation in the case of descriptions with labelled graphs, in section 4

Finally, in section 5 we give an example of the results found.

2. THE GALOIS LATTICE AS A CORRESPONDANCE BETWEEN TWO LATTICES

Using lattice theory, the formal framework is based on the use of two lattices as in (Ganascia, 1993). This model, uses lattice results given in (Birkoff, 1967) and (Barbut, 1970). This approach was used in machine learning (Ganascia, 1993), but only in propositional description.

Ganascia writes: "this framework is adequate to represent classical top-down induction systems .. but it is too restricted to formalize first order logic languages ..."

In fact, this approach can be used for structured description as well. Thus there is an unifying method for many types of description language.

2.1 Two isomorph lattices

This formalization is based on the use of two lattices: the *description lattice D* and the *instance lattice I*.

- The *instance lattice I*, corresponds to the set of parts of the *training set* ξ and is ordered by the inclusion relationship which is noted \supset where $A \supset B$ means that A is included in B. Given two elements a and b, the least upper bound - and the greatest lower bound- corresponds to the classical union -i.e \cup - and intersection -i.e \cap -.

- The *description lattice D* contains all the possible descriptions ordered by a \geq relation. This relation \geq corresponds to the generalization relationship.

$\geq : D \times D \rightarrow \{\text{true}, \text{false}\}$ for two descriptions $d_1, d_2 \in D$, $d_1 \geq d_2$ means that d_1 is more general than d_2 . Let us just consider that it structures the description space with a partial ordering.

From this relation we can define the *equivalence relation*. $\equiv : D \times D \rightarrow \{\text{true}, \text{false}\}$, $d_1 \equiv d_2$ iff ($d_1 \geq d_2$) and ($d_2 \geq d_1$)

Because D is a lattice, two elements of D have a *least upper bound*. We note $d_1 \wedge d_2$ this bound. This is the least general generalisation of d_1 and d_2 .

We have $\wedge : D \times D \rightarrow D$, if $d \geq d_1$ and $d \geq d_2$ then $d \geq d_1 \wedge d_2$.

This is a generalization operator (Plotkin, 1971), as defined in (Muggleton, 1994). For a set of description S, "A minimal generalization G of S is a generalisation of

S such that S is not a generalisation of G, and there is no generalization G' of G such that G is a generalization de G'."

2.2 Galois lattice.

Let us begin by building two correspondances between the lattice I and D.

First there is a *mapping d* between set ξ and the description space D: $d: \xi \rightarrow D$, for $e_i \in \xi$, $d(e_i) \in D$ is the description of the example e_i .

For example:

- with a propositional description, $d(e_i)$ is a list of attributes.
- in case of structural description, $d(e_i)$ can be a graph.

Now, from this simple description mapping, we can build two correspondances between I and D.

The *correspondance* $\alpha: D \rightarrow I$ associates each description d of D the set of all instances of the training set ξ which are covered by d.

$$\alpha(d) = \{e_i \in \xi \mid d \geq d(e_i)\}$$

Properties 1

- 1) $d \geq d' \Leftrightarrow \alpha(d) \supseteq \alpha(d')$
- 2) $\alpha(d_1 \wedge d_2) = \alpha(d_1) \cup \alpha(d_2)$

Proof in appendix

The *correspondance* $\beta: I \rightarrow D$ is equivalent to making the least general generalization for the description of all the elements of $H \subseteq \xi$. This means that:

$$\beta(H) = \bigwedge_{e \in H} d(e)$$

Theorem 1 The correspondance α et β defines a *Galois connection* between I and D.

Proof in appendix, see also (Ganascia, 1993).

Now we have a generalization of the classical definition of concept.

For a set of example ξ , for a description space D, for an instance space I, a *concept C* is a pair $[Ext \times Int]$ with:

- $Int \in D \mid Int = \beta(Ext) = \bigwedge_{e \in Ext} d(e)$
- $Ext \in I \mid Ext = \alpha(Int) = \{e_i \in \xi \mid Int \geq d(e_i)\}$.

All the concepts are ordered by the superconcept-subconcept (generalisation-specialisation) relation \geq_c .

$$[E_1, I_1] \geq_c [E_2, I_2] \text{ iff } E_1 \supseteq E_2 \text{ and } I_1 \supseteq I_2$$

With \geq_c , the set of all concepts has the mathematical structure of a complete lattice and is called the *Galois Lattice of the context* (ξ, d, D) .

3. DEFINITION OF THE ORDER (\geq) AND GENERALIZATION OPERATION (\wedge) FOR LABELLED GRAPHS

In section 2, we proposed a formal model. In this model we defined two basic operations \geq and \wedge . If these operations verify different properties (order, generalization operator), then the concept space is a Galois lattice.

Our goal is to use this model for structural description, more precisely for graphs descriptions.

In order to demonstrate this we must first define the operations \geq , \wedge and prove that the description space D is a lattice.

3.1 \geq Definition for labelled graphs

In this paragraph, we define an pre-order between graphs using the homomorphism relation. We will show (3.2) that for a class of graphs (core graphs), this pre-order is an order.

Notations

We note a graph $G:(V,E,L)$.

The vertex set of G is denoted by $V(G)$.

The edge set of G is denoted by $E(G)$. Each edge is a ordered pair (v_1, v_2) , $v_1, v_2 \in V(G)$.

The Label set of G is denoted by $L(G)$. For a vertex v we note $L(v)$ the label of this vertex.

In the following paragraphs, we give properties for directed graphs, these properties are true as well for undirected graphs.

Definition labelled graph homomorphism

A homomorphism $f:G_1 \rightarrow G_2$ is a mapping

$f:V(G_1) \rightarrow V(G_2)$ for which $(f(v_1), f(v_2)) \in E(G_2)$ whenever $(v_1, v_2) \in E(G_1)$ and $L(v_1)=L(v_2)$

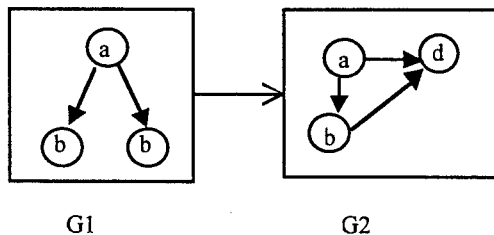


Figure 1: Homomorphism example $G_1 \rightarrow G_2$

This is not the classical subgraph isomorphism relation.

Operation \geq : $D \times D \rightarrow \{\text{True}, \text{False}\}$

For two labelled graphs $G_1:(V_1, E_1, L_1)$ and $G_2:(V_2, E_2, L_2)$, we note $G_1 \geq G_2$ iff there is a homomorphism from G_1 into G_2 .

Operation \equiv : $D \times D \rightarrow \{\text{True}, \text{False}\}$

Two labelled graphs G_1 and G_2 are *homomorphically equivalent*, denoted by $G_1 \equiv G_2$, if both $G_1 \geq G_2$ and $G_2 \geq G_1$.

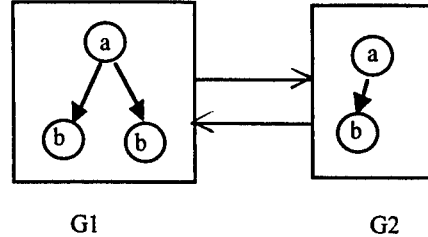


Figure 2: $G_1 \equiv G_2$

Operation \neq : $D \times D \rightarrow \{\text{True}, \text{False}\}$: $d_1 \neq d_2$ iff not ($d_1 \equiv d_2$)

3.2 D for labelled graphs

The homomorphism relation is only a pre-order because the antisymmetry property is not fulfilled (Chein, 1992). An order relation between element of D is necessary in order to use results of section 2.

The same problem occurs in Inductive Logic Programming (Muggleton, 1994)

“Because two clauses equivalent under θ -subsumption are also logically equivalent (implication), ILP systems should generate at most one clause of each equivalence class. To get around this problem, Plotkin defined equivalence classes of clauses, and showed that there is a unique representative of each clause, which he named ‘the reduced clause’”.

In the case of labelled graphs, we can use the same strategy. For this purpose, we use the class of core labelled graphs (Zhou, 1991).

Definition retract

A strict subgraph G' of G is a retract of G ((Zhou, 1991), if there is a homomorphism called a retraction $r: G \rightarrow G'$ such that $r(v)=v$ for each $v \in V(G')$.

Definition core

A graph is called a core (or minimal graph (Fellner, 1982), or irredundant graph (Cogis, 1995)) if it has no proper retracts.

Property 2

For the equivalence relation defined above (\equiv). An equivalence class of labelled graphs contains one and only one core labelled graph, which is the (unique) graph with the smallest vertex number (Mugnier, 1994).

Notation R :

We can construct a core graph from a graph as proved by Mugnier (Mugnier, 1994). This operation is called *reduction* (notation R).

Let g be a labelled graph, $R(g)$ is a core labelled graph such that $g \equiv R(g)$.

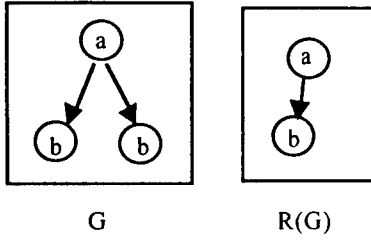


Figure 3: Example G and R(G)

We need an order relation in order to use labelled graph. For core labelled graph, we have this order. So, we define D as a set of core labelled graphs. All labelled graph description of the example can be converted to an equivalent core labelled graph, using the R operation.

Theorem 2 The restriction of \geq to the set of core labelled graphs is a lattice (Zhou, 1991), (Poole, 1993), (Chein, 1994)

For this lattice, the \vee operation is the disjoint sum of the graph, $g_1 \vee g_2 = g_1 + g_2$ (Chein, 1994) (g_1 with g_2 form a new graph).

The \wedge operation is more complex and is defined in the following paragraph.

3.3 \wedge Definition for labelled graphs

The \wedge operation for graph is based on a following classical *Kronecker product operation* \times (Weichsel, 1962).

Definition \times operation for graphs

For two graphs, the product $G_1 \times G_2$ has the vertex set $V(G_1) \times V(G_2)$ and the edges $((v_1, v_2), (v'_1, v'_2))$, where $(v_1, v'_1) \in E(G_1)$ and $(v_2, v'_2) \in E(G_2)$.

This product operation can be determined for labelled graphs.

Definition \times operation for labelled graphs

For two labelled graphs $G_1: (V_1, E_1, L_1)$ and $G_2: (V_2, E_2, L_2)$

The product $G(V, E, L) = G_1 \times G_2$ is defined by:

- $L = L_1 \cap L_2$
- $V \subseteq V_1 \times V_2 = \{ v \mid v = [v_1, v_2] \text{ with } L(v_1) = L(v_2) \text{ and } L(v) = L(v_1) \}$
- $U = \{ (v = [v_1, v_2], v' = [v'_1, v'_2]) \mid (v_1, v'_1) \in V_1 \text{ and } (v_2, v'_2) \in V_2 \} \text{ (edge oriented)}$

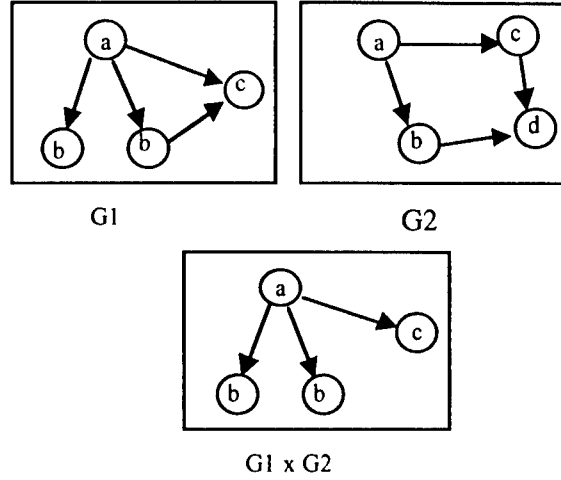


Figure 4: Labelled graphs product

Lemma 1

if G_1, G_2, G are labelled graphs then

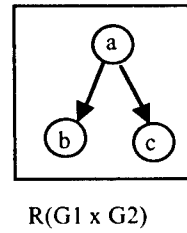
- a) $G_1 \times G_2 \geq G_1$ and $G_1 \times G_2 \geq G_2$
- b) if $G \geq G_1$ and $G \geq G_2$, then $G \geq G_1 \times G_2$
- c) $G_1 \geq G_1 \times G_2$ if and only if $G_1 \geq G_2$.

Proof: from the definitions and (Zhou, 1991)

Remark: The product operation can be easily improved when the label set is a hierarchy or a lattice.

Definition operation $\wedge: D \times D \rightarrow D$

for G_1, G_2 two core labelled graphs, $G_1 \wedge G_2 = R(G_1 \times G_2)$

Figure 5: $G_1 \wedge G_2$, with G_1 and G_2 defined in Figure 4

3.4 Galois lattice for graphs

Now, We have all the operations for the construction of a Galois lattice when example are described by graphs.

Each node of this Galois lattice is a pair $[Ext \times Int]$ with

Ext is a subset of ξ and Int is a core graph. This core graph is the generalization of the description of the examples in Ext.

Theorem 3

With:

ξ a set of examples,

D a lattice description for core labelled graphs,

d a description mapping, $d: \xi \rightarrow D$,

I the instance lattice, $\Pi(\xi)$ set power of ξ ,

\geq and order relation $D \times D \rightarrow \{\text{True}, \text{false}\}$,

\wedge the generalization operation $D \times D \rightarrow D$ for graphs.

We can define α, β .

$$\text{for } g \in D, \alpha(g) = \{e_i \in \xi \mid g \geq d(e_i)\}$$

$$\text{for } H \in I, \beta(H) = \bigwedge_{e \in H} d(e)$$

The correspondence α et β defines a Galois connection between I and D .

Proof: see lemma 1 and proof for the Theorem 1.

Using this Galois connection, we can define a Galois lattice (see 2.2). We name this lattice T .

We have defined a formal model for labelled graphs. This model uses \geq , and \wedge operations in the case of labelled graphs. In the next section, we study the complexity of these operations.

4. THE COMPLEXITY OF GALOIS LATTICE CONSTRUCTION

The complexity in the construction of a Galois lattice in our model, is a function of:

- 1) the number of nodes in the lattice,
- 2) the time and space complexity of the operations (\wedge, R) ,
- 3) the algorithm used for Galois lattice construction (see 5.1)

4.1 the size of the Galois lattice

Property 3: The number of nodes for T can be $2^{|\xi|}$.

proof: It is well known that, Galois lattice can be isomorphic to the power set of ξ (I) which is the maximal complexity for the size of T .

A similar situation occurs in our model. The proof comes from the fact that each binary attribute description can be converted to a structural description.

For example the list [Big] [Blue] [Expensive] can be structurally described as:

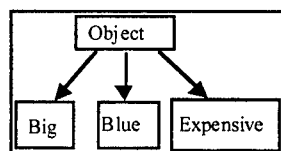


Figure 6: graph representation for an list of attributes.

Using this description, the \wedge operation for the tree representation is equivalent to \cap for the attribute representation.

4.2 Complexity of the \wedge operation

In (Muggleton, 1994) S. Muggleton and L. de Raedt wrote:

"... ILP systems can get around the problem of equivalent clauses when working with reduced clauses only".

This affirmation is true but the problem of the complexity of the R operator has not been taken into account.

- for two labelled graphs, $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, the complexity of the product is: $O(n_1 \times n_2)$ where $n_1 = |V_1|$ et $n_2 = |V_2|$.

For a set of graph P ,

$$G = \bigwedge_{G_i \in P} G_i = R(\times_{G_i \in P} G_i).$$

the size of $\times_{G_i \in P} G_i$ can be exponential.

Property 4

the operation R is co-Np-complete (Mugnier, 1994). So, in general application, this operation cannot be used.

However we do have an interesting result:

Property 5 (Mugnier, 1994)

If, for a class of labelled graphs, the homomorphism is polynomial, then the reduction operation is polynomial.

The homomorphism for the following class of labelled graphs is polynomial.

- trees (Mugnier, 1994),
- locally injective graph (Liquiere, 1994) (see definition below)
- 1/2 locally injective graph (see definition below) (see langage theory, automata (Aho, 1986))

Property 6

For a set of path or tree P , $G = \bigwedge_{G_i \in P} G_i$ is polynomial (time and size) (Horvath, 1995)

4.3 Study for a class of Graphs.

We study the complexity of the operation (\wedge, R) for the class of locally injective graphs (LIG) (Liquiere, 1994).

Notation

We note $N^+(v) = \{v' \mid (v, v') \in E\}$ and $N^-(v) = \{v' \mid (v', v) \in E\}$.

Definition LIG graph

For a labelled graph $G=(V,E,L)$, G is *locally injective* if for each vertex $v \in V$, $\forall v_1, v_2 \in N^+(v)$, $v_1 \neq v_2 \Rightarrow L(v_1) \neq L(v_2)$ and $\forall v_1, v_2 \in N^-(v)$, $v_1 \neq v_2 \Rightarrow L(v_1) \neq L(v_2)$.

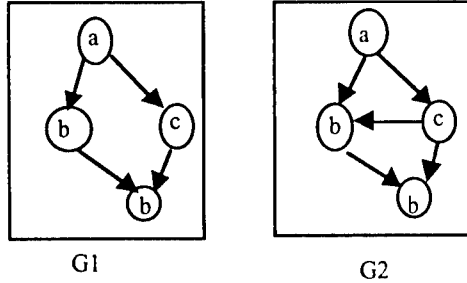


Figure 7. LIG Property

In figure 7, G_1 is an LIG graph. G_2 is not an LIG graph because, for the c node, there is two edges $c \rightarrow b$. G_2 is a 1/2 locally injective graph (see the next definition).

Definition 1/2 locally injective graph

A 1/2 locally injective graph is a oriented graph where $\forall v_1, v_2 \in N^+(v)$ (resp. $N^-(v)$), $v_1 \neq v_2 \Rightarrow L(v_1) \neq L(v_2)$.

Property 7: \geq is polynomial for locally injective graph (Liquiere, 1994) and for 1/2 locally injective graph (Aho, 1986).

Property 8: For G_1, G_2 two LIG, $G = G_1 \times G_2$ is a LIG.

Partial proof: come from the definition of \times .

Property 9: A connected LIG is an irredundant graph (Cogis, 1995)

Property 10: For G a LIG, we note $CC(G)$ the set of maximal connected subgraph of G . Then $R(G) = \{c_i \in CC(G) \mid \forall i, j, i \neq j \text{ there is no projection from } c_i \text{ to } c_j\}$

Proof: property 9 \Rightarrow Property 10

These properties are interesting because for LIG we can construct the R, \geq, \times and \wedge operations, for two graphs, with a polynomial complexity.

Property 11: For a set of 1/2 locally injective graphs P , $G = \bigwedge_{G_i \in P} G_i$ is size exponential so time exponential (results for deterministic automata (Aho, 1986)).

Table 1: Complexity for different class of language

Language	\geq, \equiv, R	Size for n graphs
Path (Horvath, 1995)	P	Polynomial
Tree (Horvath, 1995)	P	Polynomial
LIG (Liquiere, 1994)	P	?
1/2LIG (Aho, 1986)	P	Exponential
Graph (Garey, 1987)	NPC	Exponential

With P: polynomial, NPC: NP complete.

5. GRAAL IMPLEMENTATION

Traditionally machine learning offers mechanisms for a class of language. The idea is, if an algorithm is good for a general class of language, it would also work well for a less general class included in the first one. It is true, but in many cases, the general mechanism does not use all the interesting properties of the restricted language. So the complexity of the operation is not optimal for this language.

A second drawback of this approach, comes from the need for a translation process. Each description in the restricted language has to be converted into a more general one. For example a list of attributes is converted into a graph (Liquiere, 1994).

In our new method, Graal (for GRaph And Learning), we have implemented a general mechanism where description language and operations \wedge, \geq are parameters. Our tool is generic but it cannot yet be used in practical cases when important sets of examples are described by large graphs. It in fact, an algorithm for formal analysis.

5.1 A utilization of a classical Algorithm for Galois Lattice construction.

We give an algorithm which can be used on any description language with operations \leq and \wedge .

This algorithm is based on a classical method (Chein, 1969). Another algorithm can be used (Bordat, 1986) which gives the set of nodes of the Galois lattice and also the set of edges.

We note $[e_i \times d_i]$ the concept numbered i of T^k .

```

T ← ∅ /* concept set empty */
/* description of the examples */
T1 ← { {i} × d(ei) } and i ∈ [1, |ξ|]
k ← 1
While |Tk| > 1 do
  For each i < j and i, j ∈ [1, |Tk|] (so we have: [ei × di] [ej × dj]) /* we create a new concept from two concepts already found in the previous step */
    dij ← di ∧ dj /* description for the new concept */
    if dij ≠ ∅ then
      /* test if there is a concept with the same description */
      if dij ∈ Tk+1 then
        eij ← eij ∪ ej
      else
        Tk+1 ← Tk+1 ∪ [ei ∪ ej × dij]
      /* test if the description is a generalisation */
      if dij ≡ di then
        Tk ← Tk - [ei × di]
      if dij ≡ dj then
        Tk ← Tk - [ej × dj]
      End-if
    end-for
  T ← T ∪ Tk
  k ← k + 1
end-while
T ← T ∪ Tk

```

Graal is written in Java language and uses object programming properties. We have defined an abstract class (interface) so a user can add his own description language if he implements the interface.

The complexity of Galois Lattice construction with the Bordat's algorithm (Bordat, 1986) is less than $O(n^3 \cdot p)$ where n is the number of objects and p the size of T .

5.2 An experimental example.

We present an example where each object is described by a locally injective labelled graph.

We use a classical example based on arch definition.

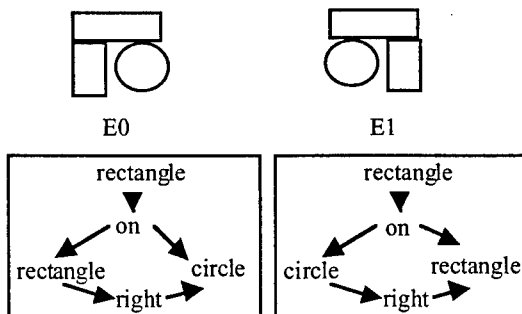


Figure 8: set of examples

The lattice is:

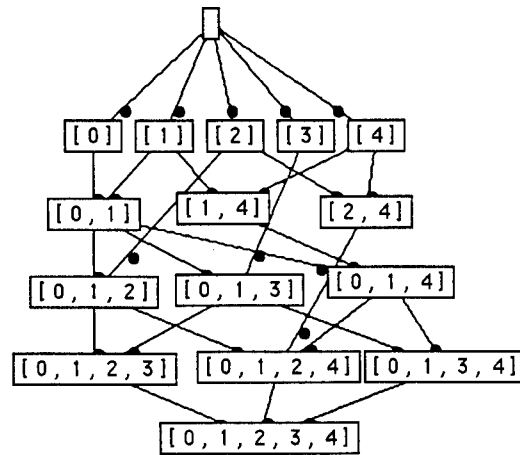


Figure 9: the structure of the Galois lattice for our set of examples.

For each node of the lattice there is a pair consisting of a graph and a set of examples. Additionally, if node₁.. node_k are linked to node_p then node_p is the least common superconcept (generalisation) of node₁.. node_k.

In figure 9 we observe the subset of examples (extension). In our tool, by double clicking on a node we obtain the following descriptions.

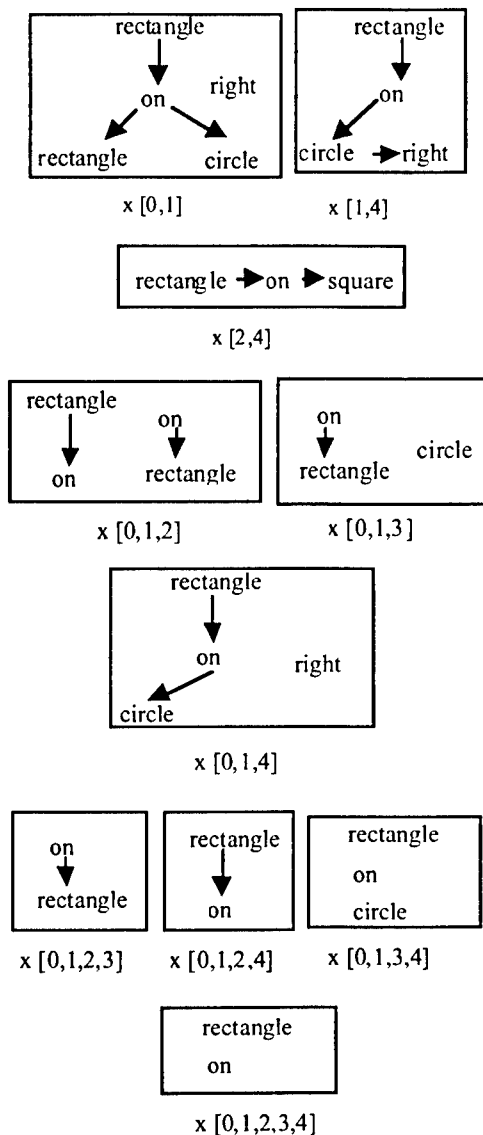


Figure 10: the graph and subset for each node.

This lattice gives all the classification for the examples without duplication. All concepts are different (description and extension), two different descriptions necessarily have distinct extensions.

Remark: For this example, the unconnected nodes like *on* can be interpreted as: there is something *on* something.

6. CONCLUSION

Our work enlarges and expands the domain of formal concept analysis by demonstrating that the Galois lattice can be used for structural description.

Coming from graph theory, our work provides operations and shows that they can be used to build a generalization operator for labelled graphs.

In addition, the LIG graphs we use are an excellent compromise between complexity and expressiveness.

Our method, written in Java, offers a general tools for formal structural concept analysis.

We are now working on the following improvements:

- To prove that LIG is PAC learnable or not,
- a survey of classical Galois lattice results in case of structural concept description,
- an implementation of heuristic in Graal, to make Graal a tool for practical application,
- an improvement of the approaches with categorical operations.

References

- F.Aho,R.Sethi,J.Ullman,"Compilers,Principles, techniques and tools",Addison wesley 1986.
- M.Barbut, B.Monjardet, 1970, " Ordre et classification: algèbre et combinatoire". Hachette,paris 70.
- Birkhoof, 1967, "Lattice theory", Third edition, American Mathematical Society, Providence , RI 1967.
- Bordat, JP 1986. "Calcul pratique du treillis de Galois d'une correspondance." Math.Sci.humaines, 24^oannée, 96:31-7.
- C.Carpineto,G.Romano,"A Lattice Conceptual Clustering System and its Application to Browsing Retrieval", Machine Learning 24,95-122 (1996).
- M.Chein,"Algorithme de recherche de sous-matrice premiere d'une matrice", bull.math.R.S.Roumanie,13, 1969.
- M.Chein, M.L Mugnier, "Conceptual Graphs: fundamental notions", RIA Volume 6- n°4, 1992,p 365-40
- M.Chein, M.L.Mugnier "Conceptual Graphs are also Graphs", Research report.
- M.C Daniel-Vatonne, C de la Higuera "Les termes: un modèle algébrique de représentation et de structuration des données", Mathématique, informatique et sciences humaines, 122, 1993, p41-63.
- T.Dietterich "unsupervised concept learning and discovery". In Readings in machine learning. pages 263-266. Morgan Kaufmann 1990.
- W.D. Fellner, "On minimal Graphs", Theoret. Comp. sci. 17 (1982) 103-110.
- J.G Ganascia, "TDis: an Algebraic Formalization", pp1008-1013, IJCAI 1993.
- B. Ganter, "Composition and Decomposition in Formal Concept Analysis", Classification and related Methods of Data analysis, p 561-566, North-Holland, Springer, NY.
- R.Godin,G.Mineau,R.Missaoui,H.Mili, " Méthode de classification conceptuelle basées sur les treillis de Galois et application ", RIA,Vol 9 n°2, 1995.
- M.R. Garey, D.S. Johnson, "Computers and Intractability: a guide to the Theory of NP-Completeness", W.H Freeman, San Francisco (USA).
- P.Hell, "Retract in graphs.", Springer Verlag Lecture notes in mathematics 406 (1974) 291-301.

O.Cogis, O.Guilnaldo, "A linear descriptor for conceptual graphs and a class for polynomial isomorphism test" pp 263-277, LNAI 954, Third conf on Concept.Struc, ICCS 1995.

T.Horvath, G.Turan, "Learning logic programs with structured background knowledge", Inductive Logic Programming conference, p53-76, Leuven, 1995.

M.Liquiere, "INNE:(Induction in Network", EWSL, Montpellier, 1989.

M.Liquiere, "Graphs and structural similarities", in New approaches in classification and data analysis. Springer Verlag. Studies in classification, data analysis. 1994.

M.Liquiere, O.Brissac, "A class of conceptual graphs with polynomial iso-projection", In proceedings of the international Conference on Conceptual Structures, ICCS, 1994.

E. Mephu Nguifo, "Galois lattice: A framework for concept learning - design, evaluation and refinement", pp 461-467, 6^o Tool with AI, New Orleans TAI 1994.

Michalski, R.S, Stepp, R "Learning from observation: conceptual clustering." In machine Learning: An artificial Approach (vol 1). Tioga publishing.

M.Missikoff, "An algorithm for insertion into a Lattice; Application to Type Classification", Proc of the 3rd Int. Conf FODO, Paris, Juin 1989, Springer Verlag, LNCS 367.

S.Muggleton, L de Raedt, "Inductive Logic Programming: Theory and Methods", J.Logic Programming 1994:19, 20:629-679.

M.L Mugnier, "On Generalisation / Specialisation for Conceptual Graphs". J.expt.theor.Artif.intell, 6, 1994.

G.D. Plotkin, "a further note on inductive generalization", in: B.Meltzer and D.Michie (eds), Machine intelligence, vol 6, Edinburgh University press, Edinburgh, 1971, 101-124.

J.Poole, J.A.Campbell, "A novel algorithm for matching conceptual and related graphs", ICCS 93, pp 293-307.

J.F.Sowa, "Conceptual structures: Information processing in mind and machine", Addison-wesley Pub., Reading, 460p, 1984.

P.M. Weichsel, "The kronecker product of graphs." Proc.Am.Math.Soc. 13 (1962) 47-52.

R.Wille "Restructuring Lattice Theory: an Approach Based on Hierarchies of concepts", in Ordered sets, I.Rival (ed), Reidel, p 445-470, 1982.

R.Wille, "Knowledge acquisition by methods of formal concept analysis", in E.Diday, editor, Data analysis, Learning Symbolic and numeric knowledge, pages 365-380. 1989.

R.Wille "Concept lattices and conceptual knowledge systems." In Computers Math.application, volume 23, pages 493-515. 1992.

H.Zhou, "Multiplicativity. Part I Variations, Multiplicative Graphs and Digraphs." Journal of graph Theory, Vol 15, N^o5, 469-488 (1991).

Appendix

Properties 1 proof

Proof 1)

$$g \geq g' \Leftrightarrow \alpha(g) \supseteq \alpha(g') \\ \Rightarrow \text{property of the order relation } \geq \\ \Leftarrow \alpha \text{ definition}$$

$$\alpha(g_1 \wedge g_2) = \alpha(g_1) \cup \alpha(g_2)$$

Proof 2)

$$\alpha(g) = \{e \in E / g \geq d(e)\}$$

$$\text{We have } g_1 \wedge g_2 \geq g_1 \text{ and } g_1 \wedge g_2 \geq g_2 \text{ so } \alpha(g_1 \wedge g_2) \\ \supseteq \alpha(g_1) \cup \alpha(g_2)$$

$$\text{We know that } g_1 \wedge g_2, \forall g, g \geq g_1 \text{ and } g \geq g_2 \text{ then } g \geq g_1 \wedge g_2. \text{ so } \alpha(g) \supseteq \alpha(g_1 \wedge g_2) \supseteq \alpha(g_1) \cup \alpha(g_2)$$

$$\text{if } \alpha(g_1 \wedge g_2) \supseteq \alpha(g_1) \cup \alpha(g_2) \text{ then for } g' / \alpha(g') = \alpha(g_1) \cup \alpha(g_2) \text{ we have}$$

$$g_1 \wedge g_2 \geq g' \text{ and } g' \geq g_1 \text{ then } g' \geq g_2 \text{ so we don't have the property of } g_1 \wedge g_2.$$

$$\text{then } \alpha(g_1 \wedge g_2) = \alpha(g_1) \cup \alpha(g_2)$$

Theorem 1 proof

α and β is a Galois connection iff :

$$a) \forall I_1 \text{ and } I_2 \in I, I_1 \subseteq I_2 \Rightarrow \beta(I_1) \leq \beta(I_2)$$

$$b) \forall g_1, g_2 \in D, g_1 \geq g_2 \Rightarrow \alpha(g_1) \supseteq \alpha(g_2)$$

$$\text{and for } h = \alpha \circ \beta \text{ and } h' = \beta \circ \alpha$$

$$c) \forall H \in I, H \subseteq h(H)$$

$$d) \forall g \in D, g \geq h'(g) \text{ (remark classically we note generalisation } \leq \text{ so we have a more classical definition.)}$$

Proof a) We have $g_1 \wedge g_2 \geq g_1$ then

$$\beta(I_1) = \bigwedge_{e \in E_1} d(e) = g \text{ and } I_1 \subseteq I_2 \beta(I_2) = (\beta(I_1)) \\ \wedge (\beta(I_2 - I_1)) \text{ so } \beta(I_2) \geq \beta(I_1)$$

Proof b) We have $g_1 \geq g_2$ and $g_2 \geq g_3 \Rightarrow g_1 \geq g_3$ because \geq is an order relation $\alpha(g_2) = \{e \in E / g_2 \geq d(e)\}$ we have $g_1 \geq g_2$ then $g_1 \geq d(e)$ with $e \in \alpha(g_2)$ so $\alpha(g_1) \supseteq \alpha(g_2)$

Proof c) We have $g_1 \geq g \Rightarrow g_1 \wedge g_2 \geq g$

$$\beta(H) = \bigwedge_{e \in H} d(e), \alpha(\beta(H)) = \{e \in E / \beta(H) \geq d(e)\}$$

But $\forall e \in H$, we have $\beta(H) \geq d(e)$ because $\beta(\{e\} \cup K) \geq d(e)$ property of \wedge .

Proof d) We have, $g \geq g_1$ and $g \geq g_2 \Rightarrow g \geq g_1 \wedge g_2$ so

$$\forall g \in D, h'(g) = \beta(\alpha(g)), \alpha(g) = \{e \in E / g \geq d(e)\},$$

$$g \geq \bigwedge_{e \in \alpha(g)} d(e) \text{ because } g \geq d(e) \text{ with } e \in \alpha(g).$$

Learning a Language-Independent Representation for Terms From a Partially Aligned Corpus

Michael L. Littman
Dept. of Computer Science
Duke University
Durham, N. C. 27708-0129
mlittman@cs.duke.edu

Fan Jiang
Dept. of Computer Science
Duke University
Durham, N. C. 27708-0129
fan@cs.duke.edu

Greg A. Keim
Dept. of Computer Science
Duke University
Durham, N. C. 27708-0129
keim@cs.duke.edu

Abstract

Cross-language latent semantic indexing is a method that learns useful language-independent vector representations of terms through a statistical analysis of a document-aligned text. This is accomplished by taking a collection of, say, English paragraphs and their translations in Spanish and processing them by singular value decomposition to yield a high-dimensional vector representation for each term in the collection. These term vectors have the property that semantically similar terms have vectors with high cosine measure, regardless of their source language. In the present work, we extend this approach to the case in which English-Spanish translations are not available, but instead, translations for documents in both languages are available in a third “bridge” language, say, French. Thus, although no aligned English-Spanish documents are used, our method creates a representation in which English and Spanish terms can be compared. The resulting vector representation of terms can be useful in natural language applications such as cross-language information retrieval and machine translation.

language generation, text comprehension and summarization, and speech recognition. This *vector lexicon* representation, in which words are “defined” by numerical vectors, has even served as a model for the acquisition of human knowledge (Landauer and Dumais, 1997).

A natural extension of the vector-lexicon representation for terms in a single language is a language-independent representation. This is a promising technique used in cross-language text retrieval (Landauer and Littman, 1990; Oard, 1997; Carbonell et al., 1997), in which natural-language queries in one language are matched against documents in another language. It may also be important to automating the creation of machine-translation systems. The key property of a good multi-lingual vector lexicon is that terms with similar meanings, regardless of their languages, are assigned vectors with high cosine measure.

A major appeal of the vector-lexicon representation is that it can be learned automatically from text. In latent semantic indexing (LSI) (Deerwester et al., 1990), this is accomplished by taking a collection of text that contains m documents (paragraphs, articles, abstracts, etc.) and n_E distinct English terms, and forming an $n_E \times m$ term-document matrix \mathcal{E} . The entry \mathcal{E}_{ij} records a value related to the number of times term i appears in document j ; in our work, we use “log entropy” weighting: $\mathcal{E}_{ij} = \ln(1.0 + \text{tf}_{ij}) \times g_i$, where

$$g_i = 1 - (\text{gf}_i \log_2(\text{gf}_i) - \sum_j (\text{tf}_{ij} \log_2(\text{tf}_{ij})) / (\text{gf}_i \log_2(m))), \quad (1)$$

tf_{ij} is the number of times term i appears in document j , and $\text{gf}_i = \sum_j \text{tf}_{ij}$. This weighting scheme gives higher weights to distinctive terms.

For any given dimensionality k ,¹ we can find a k -dimension vector lexicon by performing a singular

¹Reasonable choices for k range from 50 to 1000, de-

1 INTRODUCTION

Vector representations of word “meaning” are useful for creating computer systems that manipulate textual information. Such representations are routinely used in information retrieval (Salton and McGill, 1983; Deerwester et al., 1990) and filtering, and may find application in word sense disambiguation, natural lan-

value decomposition (SVD) and reestimating $\mathcal{E} \approx E\Sigma V^T$, where E is an $n_E \times k$ matrix, V is an $m \times k$ matrix, both E and V are orthonormal ($E^T E = I$, $V^T V = I$), and Σ is a $k \times k$ diagonal matrix of the largest singular values of \mathcal{E} . In many experiments, it has been demonstrated that the matrix E of the left singular vectors of \mathcal{E} is a useful vector lexicon of the terms. This can be justified as follows. A basic result from linear algebra (Golub and Van Loan, 1989) is that, of all rank k matrices, $E\Sigma V^T$ gives the best approximation of \mathcal{E} . Similarly, document-document correlations $\mathcal{E}^T \mathcal{E}$ are approximated by $(\Sigma V^T)^T (\Sigma V^T) \approx (\mathcal{E}^T E)(\mathcal{E}^T E)^T$. As $\mathcal{E}^T E$ is precisely the result of representing corpus \mathcal{E} via vector lexicon E , this choice of vector lexicon best captures the document-document correlations in the training corpus.

LSI has been extended to produce vector lexicons for groups of languages simultaneously by analyzing aligned document collections. Here, in addition to an $n_E \times m$ term-document matrix \mathcal{E} of English, we also have a $n_S \times m$ term-document matrix \mathcal{S} of Spanish. These matrices represent an *aligned* corpus in that the j th documents in the collections are on the same topic, or even translations of each other. Cross-language LSI (CL-LSI) (Landauer and Littman, 1990; Littman et al., 1998) works by computing a k -dimension singular value decomposition of the $(n_E + n_S) \times m$ matrix

$$\begin{bmatrix} \mathcal{E} \\ \mathcal{S} \end{bmatrix} \approx \begin{bmatrix} E \\ S \end{bmatrix} \Sigma V^T.$$

Here, the matrix $\begin{bmatrix} E \\ S \end{bmatrix}$ is orthonormal (E and S are not), and E and S can serve as vector lexicons for the English and Spanish terms, respectively.

CL-LSI has been successfully applied to many different pairs of languages, such as English-French (Littman et al., 1998), English-Japanese (Landauer et al., 1992), English-Spanish (Carbonell et al., 1997), and English-Greek (Berry and Young, 1995). It has also been applied to language triples such as English-French-Spanish and English-French-German (Rehder et al., 1997) when three-way document-aligned corpora are available.

Ultimately, our goal is to solve the *corpus hypergraph* problem, illustrated in Figure 1. In a corpus hypergraph, nodes are different languages and hyperedges represent aligned corpora. A hyperedge connects the set of languages appearing in the corresponding corpus. The corpus hypergraph problem is, given a set

pending on the application; in general, we must have $k \leq \min(m, n_E)$. We use $k = 500$ in this work.

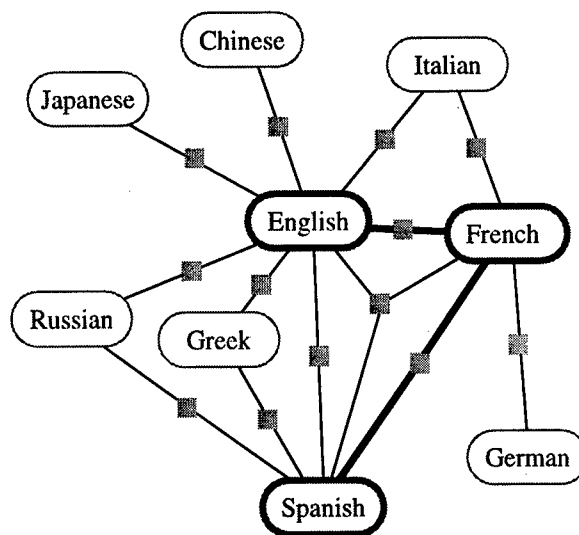


Figure 1: A corpus hypergraph shows how various languages are connected by available corpus resources. The darkened subgraph is the corpus hypergraph used in our experiments.

of languages related by corpora given in a hypergraph, support the comparison of documents expressed in any two languages connected by a path in the hypergraph. If this could be accomplished for the corpus hypergraph in Figure 1, it would become possible to compare text passages in Russian to text passages in Greek, even though no corpus was provided that related those two languages directly.

As yet, no general solution to the corpus hypergraph problem has been proposed. In this paper, we consider an important and difficult special case in which all available pairwise document-aligned corpora have a single core language in common. In our experiments, this *lingua franca* is French, although we expect English to play this role in many applications.

In particular, we consider the problem of finding a vector lexicon for English, Spanish, and French terms given a document-aligned English-French corpus and a document-aligned Spanish-French corpus. This would be represented by a corpus hypergraph with three nodes for the three languages, an edge between English and French, and another edge between Spanish and French; this is the darkened subgraph in Figure 1. We call such a corpus *partially aligned* because each document is available in only French and English or French and Spanish, but not all three languages.

Section 2 describes the way we evaluate our proposed methods. In Section 3, we show that the application

of CL-LSI to partially aligned corpora does not create adequate vector lexicons. In this method, we attempt to capture the relationships between all three languages using a single singular value decomposition. In Section 4, we show how a technique called Procrustes analysis can be applied to combine separate SVD analyses to create a unified representation. This technique shows great promise in our initial evaluations. We conclude with simple baseline comparisons and thoughts about the applicability of our approach.

2 EXPERIMENTAL EVALUATION

We will explore techniques for attacking the following problem. The input is a set of four term-document matrices: a document-aligned pair of English and French matrices, and a document-aligned pair of Spanish and French matrices. Each matrix contains 500 documents drawn from a corpus of United Nations reports (described in more detail below). The output is a 500-dimension vector lexicon defining each English, Spanish, and French term.

These vectors are then evaluated by the *mate-retrieval* test (Littman et al., 1998). In this test, we take an independent 2500-document aligned English-Spanish corpus and represent each document in the corpus by the sum of the vectors representing the terms it contains (with term vector i weighted by g_i from Equation 1). Then, for each English document vector, we compute its cosine to all 2500 of the Spanish document vectors and sort the resulting list from largest to smallest. We note the rank of the Spanish document that is the English document's translation or *mate* in the document-aligned collection. We repeat this for all 2500 English test documents and compute mean and median ranks, as well as the percentage of English documents with mates at rank 1 and ranks in the top 10.

To the extent that the vector lexicon generated is good, similar documents will get similar representations and the *mate-retrieval* test will reveal this by a low median rank. Although the *mate-retrieval* test is a poor substitute for traditional information-retrieval evaluation methods, it is sufficient for distinguishing between the techniques explored in this paper.

2.1 EXPERIMENTAL MATERIALS

Our experimental text collection was drawn from the United Nations Parallel Text Corpus (version 1.0), available through the Linguistic Data Consortium.

This collection contains approximately 1.5 gigabytes of text in English, French and Spanish. The majority of the documents were professionally translated from English, although some of the documents were originally written in Arabic, French, Spanish, Russian, or Chinese. We started with the 1990, section 00 documents in all three languages (826 files). We then removed SGML tags and extracted paragraphs with a leading number to help ensure that paragraphs were aligned between the three languages. We selected paragraphs that had at least 10 words and that varied in length no more than 75% among the three languages, yielding 6151 triplets of paragraphs, which in English ranged from 10 to 448 words (mean 68.4, s.d. 48.5).

From this collection of paragraphs, we randomly selected two disjoint sets of 500 three-way aligned paragraphs to serve as training texts. We formed 6488×500 English term-document matrices \mathcal{E} and \mathcal{E}' , 7812×500 French term-document matrices \mathcal{F} and \mathcal{F}' , and 8313×500 Spanish term-document matrices \mathcal{S} and \mathcal{S}' (\mathcal{E} , \mathcal{F} , and \mathcal{S} are aligned, and \mathcal{E}' , \mathcal{F}' , and \mathcal{S}' are aligned). Note that we tagged all terms with their source language, so our results are not due to the incidental overlap of, e.g., numerals and proper names across languages. Table 1 gives an example of the type of documents we used.

From the same collection, we extracted 2500 aligned English and Spanish test documents, $\tilde{\mathcal{E}}$ and $\tilde{\mathcal{S}}$, for use in the *mate-retrieval* experiments. All matrices are weighted using Equation 1.

3 CROSS-LANGUAGE LSI

As a first test, we applied CL-LSI to the problem of learning a vector lexicon from the partially aligned English-French-Spanish corpus.

3.1 FULLY ALIGNED CORPUS

As a baseline, we began with an experiment using the fully aligned English-French-Spanish documents. We used CL-LSI to find a 500-dimension vector lexicon for the terms in all three languages by computing an SVD of the matrix

$$\begin{bmatrix} \mathcal{E} & \mathcal{E}' \\ \mathcal{F} & \mathcal{F}' \\ \mathcal{S} & \mathcal{S}' \end{bmatrix} \approx \begin{bmatrix} E \\ F \\ S \end{bmatrix} \Sigma' V'^T,$$

yielding representations in the form of matrices E , F , and S . Evaluating these representations using the *mate-retrieval* test gave the performance listed Row 9

Table 1: An example of aligned English, Spanish, and French documents

Despite the difficult situation, there has been a strong political desire to incorporate child-related activities into the national political agenda. However, more efforts are needed to coordinate the activities of the public and private sectors. The experience gained during this period demonstrated that studies were an important contribution to the social mobilization process, and that UNICEF cooperation should further reinforce the adoption of low-cost measures for child survival and development (CSD), with greater community participation.

Pese a la difícil situación existente, se ha expresado un notable interés político por incorporar las actividades relacionadas con el niño en el programa político del país. Sin embargo, hay que desplegar mayores esfuerzos para coordinar las actividades de los sectores público y privado. La experiencia adquirida durante este período demostró que los estudios constituían una importante contribución al proceso de movilización social y que la cooperación del UNICEF debía reforzar aún más la adopción de medidas de bajo costo destinadas a la supervivencia y el desarrollo del niño en que se diera mayor participación a la comunidad.

On a pu constater qu'il existait, malgré les difficultés une forte volonté d'intégrer des activités en faveur des enfants dans le programme politique national. Mais l'action du secteur public et celle du secteur privé ne sont pas encore suffisamment coordonnées. On a aussi constaté à l'évidence durant cette période qu'il est très utile de pouvoir s'appuyer sur des études lorsqu'on cherche à mobiliser les collectivités et que dans l'action menée pour assurer la survie et le développement des enfants l'UNICEF devrait encourager encore davantage à prendre des mesures peu coûteuses et qui fassent davantage intervenir les communautés.

of Table 2. This performance is quite strong—the average rank of an English document compared to its Spanish mate is 2.0, and 99.6% of the time, the mate is ranked in the top 10. This is consistent with results obtained with CL-LSI in other text collections.

To help understand mathematically why CL-LSI is able to handle fully aligned corpora so well, consider the following idealized experiment. Imagine that $\begin{bmatrix} \mathcal{E} & \mathcal{E}' \end{bmatrix} = \begin{bmatrix} \mathcal{F} & \mathcal{F}' \end{bmatrix} = \begin{bmatrix} \mathcal{S} & \mathcal{S}' \end{bmatrix}$, as would be the case if language translation were a simple matter of word substitution (as one might get in, say, Pig Latin). In this pure case, the vector lexicon for the terms in all three languages \mathcal{E} , \mathcal{F} , and \mathcal{S} can be shown to be equal to $\sqrt{1/3}$ times the matrix of left singular vectors

of $\begin{bmatrix} \mathcal{F} & \mathcal{F}' \end{bmatrix}$. The important thing here is that the method correctly assigns identical vectors to the words in the “different” languages.

3.2 PARTIALLY ALIGNED CORPUS

In the partially aligned case, the matrices \mathcal{E}' and \mathcal{S} are not available, and we still seek to find a vector lexicon for English and Spanish so that similar terms are assigned vectors with high cosines.

We attacked this problem using CL-LSI by computing an SVD of the matrix

$$\mathcal{P} = \begin{bmatrix} \mathcal{E} & 0 \\ \mathcal{F} & \mathcal{F}' \\ 0 & \mathcal{S}' \end{bmatrix}.$$

Extracting the left singular vectors from this decomposition, we obtained a 500-dimension vector lexicon for English, French, and Spanish. The hope was that the method would be able to identify English-Spanish term relationships transitively through the common French terms. The evaluation of the derived vector lexicon using mate retrieval appears in Row 2 of Table 2.

Although this is perhaps the most elegant approach to the problem, it has the unfortunate property of giving dismal performance. In fact, the mean and median ranks are much worse than the 1250 expected by pure chance (Row 1). The reason for this poor performance is obvious in retrospect. The zeros in the definition of \mathcal{P} are meant to denote missing information (e.g., that we have no Spanish documents related to \mathcal{E} and \mathcal{F}). However, SVD treats these as real zeros and detects the fact that English and Spanish terms never co-occur; it assigns representations that capture this, making English and Spanish terms appear quite different.

A similar analysis of the idealized case from the previous section is quite instructive here. Imagine that $\mathcal{E} = \mathcal{F} = \mathcal{F}' = \mathcal{S}'$. Let F be the matrix of left singular vectors of \mathcal{F} ; this is the vector lexicon learned by LSI from an analysis of \mathcal{F} . Applying CL-LSI to \mathcal{P} yields a vector lexicon for English of $E = \begin{bmatrix} -\sqrt{1/6} F & -\sqrt{1/2} F \end{bmatrix}$ and for Spanish of $S = \begin{bmatrix} -\sqrt{1/6} F & \sqrt{1/2} F \end{bmatrix}$. This is interesting because the matrix of English-Spanish term correlations $ES^T = -1/3 FF^T$, or sign-reversed from the French-French term correlations. Essentially, terms that should have the highest similarity, like translations, are actually assigned the most dissimilar vectors by this method. While this simple analysis does not precisely capture the complexity of a realistic experiment, it does strongly suggest that CL-LSI is

Table 2: Mate-retrieval results for English to Spanish for several approaches

	METHOD	MEAN	% IN TOP 1	% IN TOP 10	MEDIAN
1	partially aligned	2177.8	0.3%	0.7%	2455
2	random order	1250.0	0.0%	0.4%	1250
3	partially aligned (reversed)	323.2	7.9%	28.9%	46
4	Procrustes on terms (to FF)	235.2	11.6%	37.8%	26
5	Procrustes on terms	216.5	11.8%	38.6%	22
6	Word matching	45.4	20.0%	47.6%	14
7	Dictionary translation	17.2	79.9%	93.8%	1
8	Procrustes on documents	14.0	57.5%	87.4%	1
9	fully aligned	2.0	92.2%	99.6%	1

not well suited to analyzing partially aligned corpora, even under idealized circumstances.

Row 3 of Table 2 gives the results of applying CL-LSI to the partially aligned corpus, then performing the mate-retrieval experiment, computing document similarities using *negative* cosine. The results here are actually quite good considering the vector lexicon is being used backwards!

In the next section, we show how to improve performance, with the added benefit that we no longer need to use the negative cosine similarity for cross-language comparisons.

4 PROCRUSTES APPROACHES

The results of the previous section show that it does not make sense to use a single SVD to create a vector lexicon from a partially aligned corpus. However, it is also clear that, within fully aligned corpora, we can use CL-LSI to find vector lexicons that produce acceptable performance.

Since a partially aligned corpus contains smaller, fully aligned corpora within it, this suggests a different strategy. Specifically, we can take the fully aligned corpus formed by \mathcal{E} and \mathcal{F} and use it to build a vector lexicons for English and French, E and F , derived by CL-LSI from a singular value decomposition of $\begin{bmatrix} \mathcal{E} \\ \mathcal{F} \end{bmatrix}$.

Once this is done, each English and French term can be thought of as a point in a high-dimensional vector space—the rows of the E and F matrices are the coordinates of the points.

The English-French vector space can also be thought to contain points for the documents in various document collections, since a document can be represented as the weighted sum of the representations of the terms

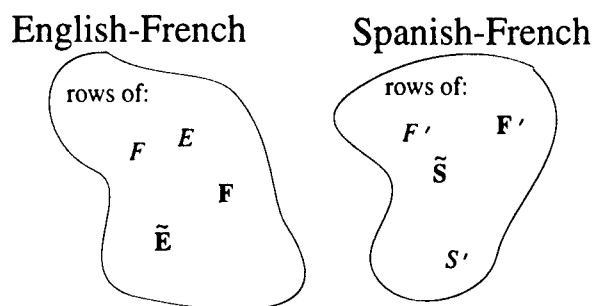


Figure 2: After performing separate CL-LSI analyses, English terms E , French terms F , English test documents \tilde{E} , and French training documents \tilde{F} occupy the English-French vector space while Spanish terms S' , French terms F' , Spanish test documents \tilde{S} , and French training documents \tilde{F}' occupy the Spanish-French vector space.

it contains. The French documents are the points corresponding to the rows of $\mathbf{F} = \mathcal{F}^T F$ and the English test documents are the points corresponding to the rows of $\tilde{\mathbf{E}} = \tilde{\mathcal{E}}^T E$. Within the English-French vector space, English terms, French terms, English documents, and French documents can be compared using the cosine metric.

Separately, we can also construct a Spanish-French vector space using CL-LSI applied to the fully aligned S' and \mathcal{F}' collections. This situation is depicted in Figure 2. Let F' and S' be the vector lexicons derived by CL-LSI; the rows of these matrices are the points corresponding to the terms in the Spanish-French vector space. The rows of $\mathbf{F}' = \mathcal{F}'^T F'$ and $\tilde{\mathbf{S}} = \tilde{\mathcal{S}}^T S'$ are the points corresponding to the French training documents and Spanish test documents, respectively.

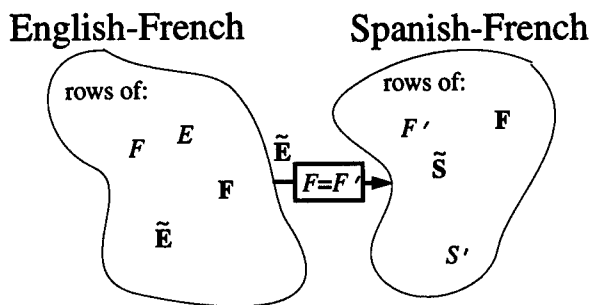


Figure 3: English term and document vectors can be transformed from the English-French vector space into the Spanish-French vector space using a Procrustes analysis derived from shared French terms.

4.1 PROCRUSTES ON TERMS

Although the two vector spaces in Figure 2 are separate, they do have something in common. Specifically, both have points corresponding to French terms. Thus, the rows of F and F' can be seen as a set of “bridge” vectors that can be used to find a way of transforming points in the English-French vector space into the Spanish-French vector space.

Mathematically, we need to derive a transformation matrix $T_{F \rightarrow F'}$ that rotates the English-French vector space so that the rows of F become as similar as possible to the rows of F' . Thus, $T_{F \rightarrow F'}$ should be a rotation matrix ($T_{F \rightarrow F'}^T T_{F \rightarrow F'} = I = T_{F \rightarrow F'} T_{F \rightarrow F'}^T$) that minimizes the Frobenius norm $\|F' - F T_{F \rightarrow F'}\|$; an algorithm for computing such a transformation (Golub and Van Loan, 1989) is given in Appendix A and is known as *Procrustes analysis*.²

Given the transformation matrix $T_{F \rightarrow F'}$, we create a new vector lexicon $\hat{E} = E T_{F \rightarrow F'}$. The vector lexicon \hat{E} has the property that English-English term similarities are unchanged from the CL-LSI-derived vector lexicon E , since

$$\begin{aligned} \hat{E} \hat{E}^T &= E T_{F \rightarrow F'} (E T_{F \rightarrow F'})^T \\ &= E T_{F \rightarrow F'} T_{F \rightarrow F'}^T E^T = E E^T; \end{aligned}$$

however, these English term vectors now occupy the Spanish-French vector space, so comparisons between English and Spanish terms can be made. Figure 3 illustrates this procedure.

To evaluate this idea, we carried out the following experiment. First, we listed all the French terms that

²Procrustes is the robber of Greek legend who forced people of varying sizes to fit perfectly in a fixed sized bed by stretching or cutting them as appropriate.

appear in both \mathcal{F} and \mathcal{F}' (for the purposes of this experiment, we excluded terms with digits in them). This results in a set of 3311 French terms, from which we picked 500 terms uniformly at random without replacement to form a set of bridge vectors.³ We used these terms to find a transformation from English-French vector space to Spanish-French vector space, applied this transformation to the English test documents $\tilde{E} T_{F \rightarrow F'}$, and performed the mate-retrieval test against the Spanish test documents \tilde{S} . The result of this test is given in the Row 5 of Table 2.

The median rank of mates for this method is 22, making it better than the performance of “reversed” CL-LSI applied to this partially aligned corpus (median rank of mate of 46), but still far from the performance of CL-LSI on the fully aligned corpus (median rank of mate of 1). Note that transforming the Spanish terms into the English-French vector space via an appropriately defined $T_{F' \rightarrow F}$ gives precisely the same mate-retrieval performance because of the symmetric nature of the transformation matrix.

To understand why Procrustes on terms did not perform quite up to expectations, we tested a fundamental hypothesis behind this approach. In particular, we were assuming that the French vector lexicon derived from the English-French corpus was essentially the same as that derived from the Spanish-French corpus. In particular, we were assuming that the French term-term correlations in the two vector spaces were roughly the same. To test whether this was true, we measured the stability of the term-term similarities in the two vector spaces by the correlation between $F F^T$ and $F' F'^T$; it is 0.52, suggesting a positive correlation, but perhaps not one strong enough to form an ideal bridge between English and Spanish.

4.2 PROCRUSTES ON DOCUMENTS

We felt we could establish a stronger bridge using a more stable French vector lexicon. We considered the corpus formed by taking the union of the documents in \mathcal{F} and \mathcal{F}' . Let F_{FF} be the vector lexicon derived by LSI on the matrix $\begin{bmatrix} \mathcal{F} & \mathcal{F}' \end{bmatrix}$. This vector lexicon defines a French-French vector space that contains French terms F_{FF} as well as both sets of French training documents $F_{FF} = \mathcal{F}^T F_{FF}$ and $F_{FF}' = \mathcal{F}'^T F_{FF}$.

As we had hoped, this procedure strengthened somewhat the stability of the French vector lexicons. In

³Of course, there are many other ways one might select a set of terms to create bridge vectors. Our choice was motivated by a combination of simplicity and practicality.

particular, for the set of French bridge terms, the correlation between FF^T and $F_{FF}F_{FF}^T$ is 0.74 and the correlation between $F'F'^T$ and $F_{FF}F_{FF}^T$ is 0.72. Hence, we'd expect the bridge between the English-French vector space and the French-French vector space to be stronger than the one between the English-French vector space and the Spanish-French vector space from the previous experiment, where the correlation was 0.52.

To evaluate the revised method, we constructed $T_{F \rightarrow F_{FF}}$ and $T_{F' \rightarrow F_{FF}}$ by Procrustes analysis and transformed the English and Spanish test documents from their respective vector spaces into the French-French vector space.

Performing a mate-retrieval test on these collections gave the results appearing in Row 4 of Table 2, which are just slightly worse than those of the previous experiment (e.g., median rank of mate of 26 instead of 22), in spite of the apparently higher correlations for the bridge vectors. We explain this by noting that in the previous experiment, English terms were transformed into the Spanish-French vector space via a transformation with a correlation of 0.52. Here, before English and Spanish terms or documents can be compared, *two* transformations take place. Under an independence assumption, we estimate the cumulative correlation of the two transformations as $.74 \times .72 \approx 0.53$, which is about the same as in the previous experiment.

We then repeated the experiment with the difference that we based our transformations on document vectors instead of term vectors. Our intuition is that documents are more stable predictors of meaning than are individual terms.

Figure 4 illustrates our technique of merging vector spaces by a Procrustes analysis of shared documents. The basic idea is that we first create separate English-French, Spanish-French, and French-French vector spaces from the various combinations of pieces of the partially aligned corpus. We then create vector representations for the French training documents \mathcal{F} in both the English-French \mathbf{F} and French-French \mathbf{F}_{FF} spaces. This allows us to create a transformation matrix $T_{F \rightarrow F_{FF}}$ from English-French to French-French via Procrustes analysis. English test documents $\tilde{\mathbf{E}}$ can now be located in the French-French space via

$$\tilde{\mathbf{E}} T_{F \rightarrow F_{FF}}.$$

A similar analysis results in transforming the Spanish test documents into the French-French vector space.

The resulting vectors for the test documents can then

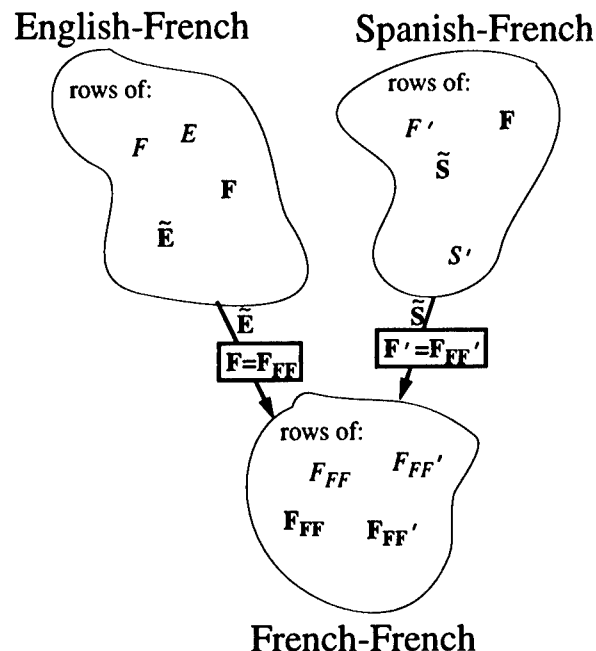


Figure 4: English term and document vectors can be transformed from the English-French vector space into the French-French vector space using a rotation derived from shared French documents. Spanish term and document vectors can be transformed analogously.

be evaluated using the mate-retrieval test. The performance, given in Row 8 of Table 2, is quite encouraging. We find that, more than half the time English test documents are closest to their Spanish mates and 87.4% of the time, the mate is ranked in the top 10 out of 2500 Spanish test documents. This is not quite as good as the results of training directly on aligned English-Spanish documents, but, considering the small amount of training data, the performance is quite respectable.

It is instructive to look once more at the correlations of the bridge vectors used to map between the various vector spaces. The correlation between the French \mathcal{F} document-document correlations in the English-French vector space $\mathbf{F}\mathbf{F}^T$ and the French-French vector space $\mathbf{F}_{FF}\mathbf{F}_{FF}^T$ is 0.98. Similarly, the correlation between the French \mathcal{F}' document-document correlations in the Spanish-French vector space $\mathbf{F}'\mathbf{F}'^T$ and the French-French vector space $\mathbf{F}_{FF}'\mathbf{F}_{FF}'^T$ is 0.98. The product of these correlations is 0.97, providing strong evidence that the transformations that bring the English and Spanish test documents together in the French-French space are robust and accurate.

5 BASELINE COMPARISONS

To help evaluate our results, we carried out two simple comparative experiments. First, we repeated the mate-retrieval evaluation using English to retrieve Spanish using only direct word matching. That is, only words appearing in both languages (proper names, numbers, acronyms, and some cognates like "idea") were used to measure the similarity between documents. This type of information is routinely used to align texts and it is reasonable to ask how this method compares to Procrustes by documents.

The result of mate-retrieval via word matching appears in Row 6 of Table 2. This method scores between the term-based Procrustes methods and Procrustes by documents. Thus, Procrustes by documents is extracting useful information beyond simply pairing up cross-language homographs.

A second comparison used an available 21,000-word English-Spanish bilingual dictionary to compare English and Spanish documents. Although this approach does not construct a vector lexicon, it does help establish the baseline difficulty of the mate-retrieval task when additional linguistic resources are available.

We created a simple word-for-word translation system from English to Spanish. For each English word found in the dictionary, we substituted all Spanish entries. English words not found were left untranslated (allowing proper names and acronyms to pass through). We performed no stemming or morphological analysis in either language. The translation procedure substituted about 23.6% of types and 69% of tokens in the original English.

The translated English documents were then matched against the Spanish test documents and mate-retrieval statistics collected; they are given in Row 7 of Table 2. In terms of mean rank of mate, Procrustes on documents, which used no direct English-Spanish information, performed better than the human constructed dictionary (14.0 vs. 17.2). However, dictionary translation scored better in percentage in top 1 and top 10.

The results in Table 2 suggest that, when a parallel corpus is available (Row 9), it is the most effective method for constructing a vector lexicon. In the absence of a parallel corpus, a bilingual dictionary can be used to match documents with one another and this works relatively well (Row 7), at least for "query by example" type of tasks. Procrustes on documents (Row 8) performs a bit less well by some measures, but does not require any direct resources relating the

languages in question.

That a parallel corpus technique outperforms dictionary translation is consistent with earlier studies measuring average precision for information-retrieval tasks (Carbonell et al., 1997). We are in the process of evaluating Procrustes by documents on this task.

6 CONCLUSIONS

In this paper, we introduce the problem of learning multi-lingual vector lexicons for partially aligned corpora. We couch the general problem in terms of taking a corpus hypergraph and finding vector representations for all the terms so that semantically similar terms have representations with high cosine, independent of language.

Although we do not propose a solution to the general corpus hypergraph problem, we attack the specific case in which two languages are related only indirectly through a third language. Specifically, we have an English-French aligned corpus and a Spanish-French aligned corpus and we want to make comparisons transitively between English and Spanish terms. We evaluate several algorithms for deriving vector lexicons from this type of corpus.

Of the approaches we considered, the most successful was Procrustes on documents, in which English and Spanish terms are transformed into a French-only vector space on the basis of the representation of a core set of French documents. A vector lexicon derived by this method was able to perfectly identify Spanish translations of English documents from a field of 2500 choices 57.5% of the time. While this is not nearly as accurate as a vector lexicon trained directly on aligned English and Spanish documents (92.2%), it is encouraging given that all English-Spanish term-term relationships were derived indirectly and completely automatically through connections with French terms.

Our technique is applicable for finding relationships among arbitrarily large sets of languages as long as all these languages are related through aligned corpora with some core language (be it French, English, Russian, etc.). In future work, we hope to extend this approach to use information in a web of aligned corpora to establish a robust representation for terms in any of the world's languages.

Acknowledgments

We thank Tom Landauer for suggesting the idea of using Procrustes analysis, Sue Dumais and Mike Berry for pointing us to the definition, Steve Majercik and Xiaobai Sun for helping us understand its derivation, and Dian Martin for early encouraging results.

A PROCRUSTES DERIVATION

We seek a transformation $T_{A \rightarrow B}$ that rotates the vectors in the rows of A and makes them as similar as possible to the vectors of the rows of B . Specifically, we want $T_{A \rightarrow B}$ to be the $k \times k$ orthonormal matrix Q such that Frobenius norm $\|B - AQ\|$ is minimized over all matrices Q with $Q^T Q = I = Q Q^T$.

First, using the definition that $\text{trace}(X)$ is the sum of the diagonal entries of X and $\|X\|$ is the sum of the squares of all the entries of X ,

$$\begin{aligned} \min_{Q^T Q = I} \|B - AQ\| &= \min_{Q^T Q = I} \text{trace}((B - AQ)(B - AQ)^T) \\ &= \min_{Q^T Q = I} (\text{trace}(BB^T) - \text{trace}(BQ^T A^T) \\ &\quad - \text{trace}(AQB^T) + \text{trace}(AA^T)) \\ &= \max_{Q^T Q = I} \text{trace}(AQB^T). \end{aligned}$$

Now, note that $\text{trace}(XY) = \text{trace}(YX)$ and let $A^T B = U \Sigma V^T$ be the singular value decomposition of $A^T B$. This gives us

$$\begin{aligned} \max_{Q^T Q = I} \text{trace}(AQB^T) &= \max_{Q^T Q = I} \text{trace}(B^T A Q) \\ &= \max_{Q^T Q = I} \text{trace}(V \Sigma U^T Q) \\ &= \max_{Q^T Q = I} \text{trace}(\Sigma U^T Q V). \end{aligned}$$

Since V , Q , and U are all orthonormal, they cannot increase the length of any of the rows of Σ . This means that the maximum of the sum of the diagonal entries of $\text{trace}(\Sigma U^T Q V)$ occurs when $U^T Q V = I$, or when $Q = UV^T$. Thus, $T_{A \rightarrow B} = UV^T$ is the transformation that maximizes the alignment between A and B .

References

- Berry, M. W. and Young, P. G. (1995). Using Latent Semantic Indexing for multilanguage information retrieval. *Computers and the Humanities*, 29(6):413-429.
- Carbonell, J., Yang, Y., Frederking, R., Brown, R. D., Geng, Y., and Lee, D. (1997). Translingual information retrieval: A comparative evaluation. In *Proceedings of Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391-407.
- Golub, G. H. and Van Loan, C. F. (1989). *Matrix Computations*. The Johns Hopkins University Press, 2 edition.
- Landauer, T. K. and Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104(2):211-240.
- Landauer, T. K. and Littman, M. L. (1990). Fully automatic cross-language document retrieval using latent semantic indexing. In *Proceedings of the Sixth Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research*, pages 31-38. Waterloo Ontario.
- Landauer, T. K., Littman, M. L., and Stornetta, W. S. (1992). A statistical method for cross-language information retrieval. Unpublished manuscript.
- Littman, M. L., Dumais, S. T., and Landauer, T. K. (1998). Automatic cross-language information retrieval using latent semantic indexing. In Grefenstette, G., editor, *Cross Language Information Retrieval*. Kluwer.
- Oard, D. W. (1997). Alternative approaches for cross-language text retrieval. In Hull, D. and Oard, D., editors, *Cross-Language Text and Speech Retrieval: Papers from the 1997 AAAI Spring Symposium*. The AAAI Press.
- Rehder, B., Littman, M. L., Dumais, S., and Landauer, T. K. (1997). Automatic 3-language cross-language information retrieval with latent semantic indexing. In *The Sixth Text Retrieval Conference Notebook Papers (TREC6)*, pages 103-110. National Institute of Standards and Technology Special Publication.
- Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.

Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes

John Loch

Department of Computer Science
University of Colorado
Boulder, CO 80309-0430
loch@cs.colorado.edu

Satinder Singh

Department of Computer Science
University of Colorado
Boulder, CO 80309-0430
baveja@cs.colorado.edu

Abstract

Recent research on hidden-state reinforcement learning (RL) problems has concentrated on overcoming partial observability by using memory to estimate state. However, such methods are computationally extremely expensive and thus have very limited applicability. This emphasis on state estimation has come about because it has been widely observed that the presence of hidden state or partial observability renders popular RL methods such as Q-learning and Sarsa useless. However, this observation is misleading in two ways: first, the theoretical results supporting it only apply to RL algorithms that do not use eligibility traces, and second these results are worst-case results, which leaves open the possibility that there may be large classes of hidden-state problems in which RL algorithms work well without any state estimation.

In this paper we show empirically that Sarsa(λ), a well known family of RL algorithms that use eligibility traces, can work very well on hidden state problems that have good memoryless policies, i.e., on RL problems in which there may well be very poor observability but there also exists a mapping from immediate observations to actions that yields near-optimal return. We apply conventional Sarsa(λ) to four test problems taken from the recent work of Littman, Littman, Cassandra and Kaelbling, Parr and Russell, and Chrisman, and in each case we show that it is able to find the best, or a very good, memoryless policy without any of the computational expense of state estimation.

1 Introduction

Sequential decision problems in which an agent's sensory observations provide it with the complete state of its environment can be formulated as Markov decision processes, or MDPs, for which a number of very successful planning (Sutton & Barto, 1998) and reinforcement learning (Barto et al., 1983; Sutton, 1988; Watkins, 1989) methods have been developed. However, in many domains, e.g., in mobile robotics, and in multi-agent or distributed control environments, the agent's sensors at best give it partial information about the state of the environment. Such agent-environment interactions suffer from hidden-state (Lin & Mitchell, 1992) or perceptual aliasing (Whitehead & Ballard, 1990; Chrisman, 1992) and can be formulated as *partially observable Markov decision processes*, or POMDPs (e.g., Sondik, 1978). Therefore, finding efficient reinforcement learning methods for solving interesting sub-classes of POMDPs is of great practical interest to AI and engineering.

Recent research on POMDPs has concentrated on overcoming partial observability by using memory to estimate state (Chrisman, 1992; McCallum, 1993; Lin & Mitchell, 1992) and on developing special purpose planning and learning methods that work with the agent's state of knowledge, or belief state (Littman et al., 1995). In part, this emphasis on state estimation has come about because it has been widely observed and noted that the presence of hidden state renders popular and successful reinforcement learning (RL) methods for MDPs, such as Q-learning (Watkins, 1989) and Sarsa (Rummery & Niranjan, 1994), useless on POMDPs (e.g., Whitehead, 1992; Littman, 1994; Singh et al., 1994). However, this observation is misleading in two ways: first, the theoretical results (Singh et al., 1994; Littman, 1994) supporting it only apply to RL algorithms that do not use eligibility

traces, and second, these results are worst-case results which leaves open the possibility that there may be large classes of POMDPs in which existing RL algorithms work well without any state estimation.

The main contribution of this paper is to show empirically that Sarsa(λ), a well known family of reinforcement learning algorithms that use eligibility traces, can work very well on POMDPs that have good memoryless policies, i.e., on problems in which there may well be very poor observability but there also exists a mapping from the agent's immediate observations to actions that yields near-optimal return. We also show how this can be extended to low-order-memory-based policies. This contribution is significant, because it may be that most real-world engineering problems that are well designed have good memoryless or good low-order-memory-based policies. We apply conventional Sarsa(λ) on four test problems taken from recent published work on POMDPs and in each case show that it is able to find the best, or a very good, memoryless policy without any of the computational expense of state estimation. However, these results have to be interpreted with caution for the problem of finding optimal memoryless policies in POMDPs is known to be computationally challenging (Littman, 1994); they are evidence that Sarsa(λ) is at least competitive to and at best better than other existing algorithms for solving POMDPs when good low-order-memory-based policies exist.

2 POMDP Framework

In this section we briefly describe the POMDP framework. An environment is defined by a finite set of states S , the agent has recourse to a finite set of actions A , and the agent's sensors provide it observations from a finite set X . On executing action $a \in A$ in state $s \in S$ the agent receives expected reward r_s^a and the environment transits to a random state $s' \in S$ with probability $P_{ss'}^a$. The probability of the agent observing $x \in X$ given that the environment's state is s is $O(x|s)$. In the reinforcement learning (RL) problem the agent does not know the transition and observation probabilities P and O and its goal is to learn an action selection strategy that maximizes the *return*, i.e. the expected discounted sum of rewards received over an infinite horizon, $E\{\sum_{t=0}^{\infty} \gamma^t r_t\}$, where $0 \leq \gamma < 1$ is the discount factor that makes immediate reward more valuable than reward more distant in time, and r_t is the reward at time step t .

In fully observable RL problems or MDPs it is known

that there exists an optimal policy that is memoryless, i.e., is a mapping from states to actions, $S \rightarrow A$. RL algorithms such as Q-learning and Sarsa are able to provably find such memoryless optimal policies in MDPs. It is known that in POMDPs the best memoryless policy can be arbitrarily suboptimal in the worst case (Singh et al., 1994). We ask below if these same RL algorithms can find the best memoryless policy in POMDPs (Jaakkola et al., 1995; Littman, 1994), regardless of how good or how bad it is; for if they are able to find it, then they can at least be useful in POMDPs with good memoryless policies. We note that the success of RL algorithms when using compact function approximation in fully observable problems (Barto et al., 1983; Tesauro, 1995) provides some evidence that this is possible because the use of compact function approximation introduces hidden state into otherwise completely observable MDPs.

3 Eligibility Traces and Sarsa(λ)

In MDPs reinforcement learning algorithms such as Sarsa(λ) use experience to learn estimates of optimal Q-value functions that map state-action pairs, s, a , to the optimal return on taking action a in state s . The transition at time step t , $\langle s_t, a_t, r_t, s_{t+1} \rangle$, is used to update the Q-value estimate of all state-action pairs in proportion to their eligibility. The idea behind the eligibilities is very simple. Each time a state-action pair is visited it initiates a short-term memory or trace that then decays over time (exponentially with parameter $0 \leq \lambda \leq 1$). The magnitude of the trace determines how eligible a state-action pair is for learning. So state-action pairs visited more recently are more eligible.

In POMDPs the transition information available to the agent at time step t is $\langle x_t, a_t, r_t, x_{t+1} \rangle$. A straightforward way to extend RL algorithms to POMDPs is to learn Q-value functions of observation-action pairs, i.e., to simply treat the agent's observations as states. Below we describe standard Sarsa(λ) applied to POMDPs in this manner. At step t the Q-value function is denoted Q_t and the eligibility trace function is denoted η_t . On experiencing transition $\langle x_t, a_t, r_t, x_{t+1} \rangle$ the following updates are performed in order:

$$\begin{aligned} \eta_t(x_t, a_t) &= 1 \\ \forall (x \neq x_t \text{ or } a \neq a_t); \eta_t(x, a) &= \gamma \lambda \eta_{t-1}(x, a) \\ \forall x \text{ and } a; \\ Q_{t+1}(x, a) &= Q_t(x, a) + \alpha * \delta_t * \eta_t(x, a) \quad (1) \end{aligned}$$

where $\delta_t = r_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)$, and α is the step-size. The eligibility traces are initialized to zero, and in episodic tasks they are reinitialized to zero after every episode. The greedy policy at time step t assigns to each observation x the action $a = \operatorname{argmax}_b Q_t(x, b)$. Note that the greedy policy is memoryless.

3.1 Using Sarsa(λ) with Observation Histories

The Sarsa(λ) algorithm can also be easily used to develop memory-based policies by simply learning a Q-value function over estimated-states and actions, and by keeping eligibility traces for estimated-state and action pairs. So for example, we could augment the immediate observation with the past K observations to form the estimated-state and derive a memory-based policy that maps $K + 1$ observations to actions. The only change to the equations in (1) would be that the immediate observations (x 's) would be replaced by the estimated states.

4 Empirical Results

The Sarsa(λ) algorithm was applied in an identical manner to four POMDP problems taken from the recent literature and described below. Here we describe the aspects of the empirical results common to all four problems. At each step, the agent picked a random action with a probability equal to the exploration rate, and a greedy action otherwise. Except where explicitly noted, we used an initial exploration rate of 20% decreasing linearly with each action (step) until the 200000th action from where onwards the exploration rate was 0%. Q-values were initialized to 0. The agent starts each episode in a problem specific start state or a randomly selected start state as specified by the originators of the problems. Both the step-size (α) and the λ values are held constant in each experiment. We did a coarse search over α and λ for each problem but present results only for $\lambda = 0.9$ and $\alpha = 0.01$ which gave about the best performance across all problems. In all cases, a value of λ between 0.8 and 0.975 worked the best. This is qualitatively similar to the results obtained for MDPs, and a bit surprising given that Sarsa(1) (or Monte-Carlo) has been recommended as the way to deal with hidden state (Singh et al., 1994).

The data for the learning curves is generated as follows: after every 1000 steps (actions) the greedy policy is evaluated offline to generate a problem specific performance metric. All the learning-curves below are

plotted after smoothing this data by doing a running average over 30 data points.

For each POMDP we first present its structure by defining the states, actions, rewards, and observations and then we present our results.

4.1 Sutton's Grid World

Sutton's grid world problem (see Figure 1A) is from Littman (1994) who took a navigation gridworld from Sutton (1990) and made it a POMDP by not allowing its exact position to be known to the agent.

States: This POMDP is a 9 by 6 grid with several obstacles and a goal in the upper right corner (see Figure 1A). The state of the environment is determined by the grid square the agent occupies. State transitions are deterministic.

Actions: The agent can choose one of 4 actions: move north, move south, move east, and move west.

Observations: The agent can observe its 8 neighboring grid squares yielding 256 possible observations. Only 30 (of the 256 possible) unique observations occur in the gridworld. Observations are deterministic. Figure 1A shows the gridworld with observations indicated by the number in the lower right corner of each square.

Rewards: The agent receives a reward of -1 for each action that does not transition to the goal state. A reward of 0 is received for any action leading to the goal state.

When the agent reaches the goal state it transitions to a uniformly random start state.

4.1.1 Sarsa(λ) Results

After every 1000 steps of experience in the world, the greedy policy is evaluated to determine the total number of steps required to reach the goal from every possible non-goal start state (46 start states). The agent is limited to a maximum of 1000 steps to reach the goal. Thus a policy which cannot reach the goal from any start state would have a total steps to goal of 46,000.

Sarsa(λ) converged to the 416 total step policy shown with arrows in Figure 1A; the learning-curve is shown in Figure 1B. The total steps to the goal for the optimal policy in the underlying MDP is 404, and so in this case a very good memoryless policy was found. This 416 step policy matches exactly with the 416 step policy Littman (1994) found using an expensive branch

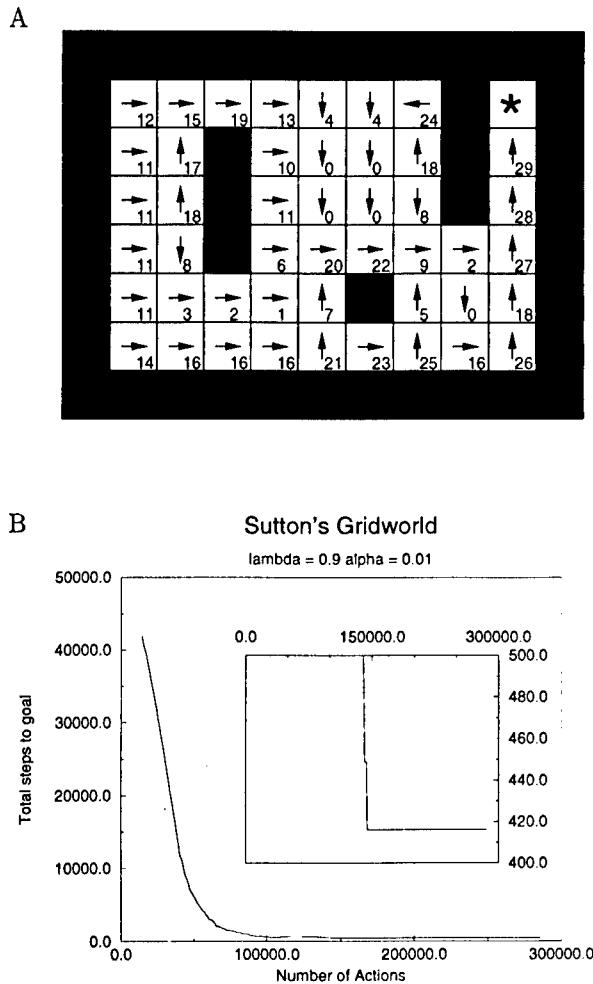


Figure 1: Sutton's Grid World (from Littman, 1994). A) The grid world environment. The numbers on the lower right are the observations. The arrows show the optimal memoryless policy found by Sarsa. B) The total steps to goal of the greedy policy as a function of the amount of learning steps. The inset plot shows the same data at a different scale.

and bound method that searches directly in memoryless policy space and is guaranteed to find the optimal memoryless policy. Note that the number of possible memoryless policies is $(4 \text{ actions}, 30 \text{ observations}) 4^{30} = 1.2 \times 10^{18}$ policies.

Observe that in Figure 1A the agent learns to go left in the state just to the left of the goal. This is because it has to go up in the state immediately below (observation 18) because of its aliasing with the state 4 steps below the goal (both states have 3 walls to the right).

4.2 Littman, Cassandra, and Kaelbling's 89 State Office World

States: The gridworld for Littman et al.'s (1995) 89 state office problem is shown in Figure 2A. The state of the environment is the combination of the grid square that the agent is occupying and the direction that the agent is facing (N, S, E, W). There are 22 possible agent locations times 4 directions for 88 states plus the goal state for 89 total states. State transitions are stochastic.

Actions: The agent can choose one of 5 actions: stay in place, move forward, turn right, turn left, and turn around. Both the state transitions and the observations are noisy with the agent getting the correct observation only 70% of the time.

Observations: The agent can observe the relative position of obstacles in 4 directions: front, back, left and right. There are 16 possible observations plus the goal observation.

Rewards: The agent receives a reward of +1 for any action leading to the goal observation with all other rewards equal to 0.

After reaching the goal observation the agent transitions to a uniformly random start state.

4.2.1 Sarsa(λ) Results

After every 1000 steps of experience the greedy policy is evaluated. As in Littman et al., for each evaluation 251 trials are run using the greedy policy with a maximum step cutoff at 251 steps. Two performance metrics are used: the median number of steps to the goal for the 251 trials, and the percent of the 251 trials which reach the goal state within 251 steps.

The best memoryless policy found by Sarsa(0.9) was able to reach the goal on average 77% of the 251 trials (see Figure 2B) with a median number of steps to goal of 73 steps (see Figure 2C). The best policy

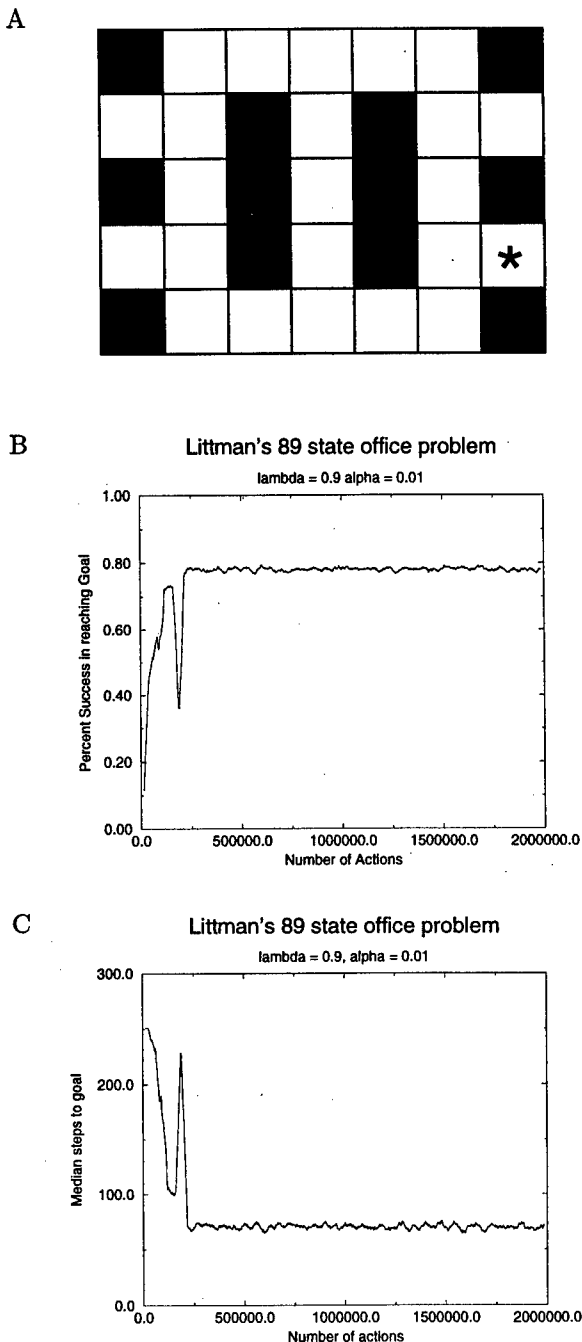


Figure 2: Littman et al.'s 89 state office world. A) The office world environment where the goal state is denoted with a star. The state of the environment is the combination of the grid square that the agent is occupying and the direction that the agent is facing (N, S, E, W). B) The percentage of trials with the greedy policy that succeed in getting to the goal in less than 251 steps. C) Median number of steps to goal of the greedy policy as a function of the number of learning steps.

found by Sarsa(0.9) outperformed all of the memory-based policies found by Littman et al. in their Table 3. Their best policy was able to reach the goal in only 44.6% of the 251 trials with a median steps to goal of > 251 steps and was found using truncated value iteration algorithm on belief states. Littman et al. also presented a hybrid method that finds a policy that reached the goal in 58.6% of the trials (still below the percentage for the best memoryless policy found by Sarsa(0.9) with median steps to goal of 51 steps (this is better than Sarsa(0.9)'s 73 steps).

There are $5^{16} = 1.53 \times 10^{11}$ possible memoryless policies for this problem. Therefore it is not practical to enumerate the performance of every possible policy to verify if the policy found by Sarsa(0.9) is indeed the optimal memoryless policy, but its performance vis-a-vis the state-estimation based methods of Littman et al. was encouraging.

4.3 Parr and Russell's Grid World

States: Parr and Russell's (1995) gridworld consists of 11 states in a 4 by 3 grid with a single obstacle (see Figure 3A). The state of the environment is determined by the grid square occupied by the agent.

Actions: The agent can choose one of 4 actions: move north, move south, move east, and move west. State transitions are stochastic with the agent moving in the desired direction 80% of the time and slipping to either side 10% of the time.

Observations: The agent can only observe if there is a wall to its immediate left or right. There are 4 possible observations corresponding to the combinations of left and right obstacles plus two observations for the goal and penalty states yielding a total of 6 observations. Observations are deterministic.

Rewards: There is a goal state in the upper right corner with a penalty state directly below the goal state. The agent receives a reward of -0.04 for every action which does not lead to the goal or penalty state. The agent receives a reward of $+1$ for any action leading to the goal state and a reward of -1 for any action leading to the penalty state.

4.3.1 Sarsa(λ) Results

Every 1000 steps the greedy policy was evaluated and the learning curve is presented in Figure 3B. The average reward per step was computed for 101 trials of up to 101 steps per trial. There are $4^6 = 4096$ possible memoryless policies for this problem. We veri-

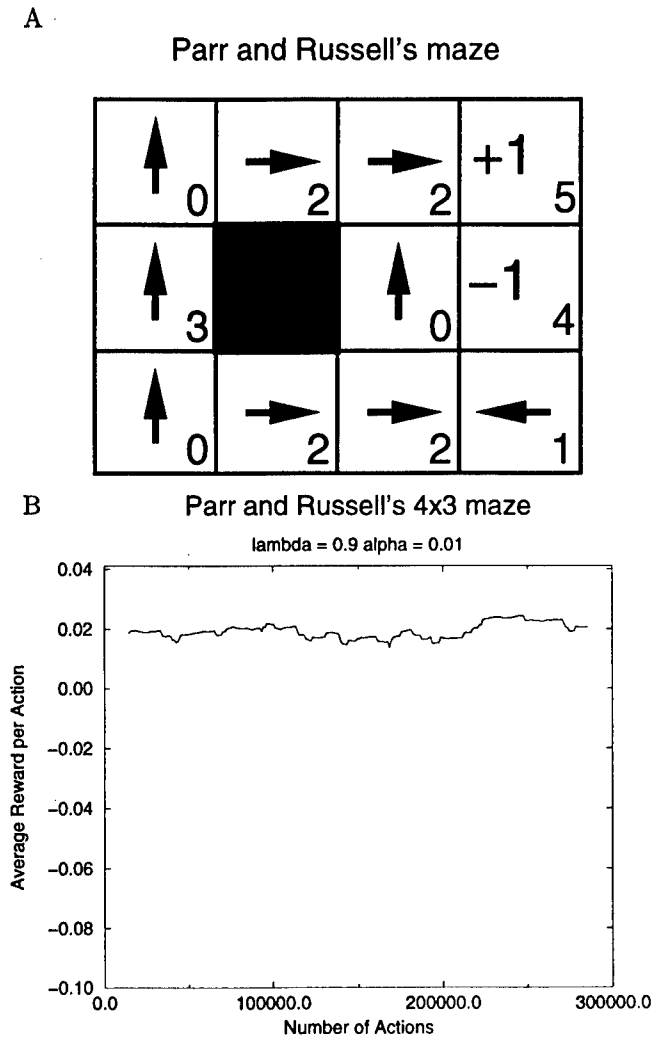


Figure 3: Parr & Russell's Grid World. A) The grid-world environment. The numbers in the lower right are the observations. The arrows show the optimal memoryless policy found by Sarsa. B) The average reward per action of the memoryless greedy policy as a function of the number of learning steps.

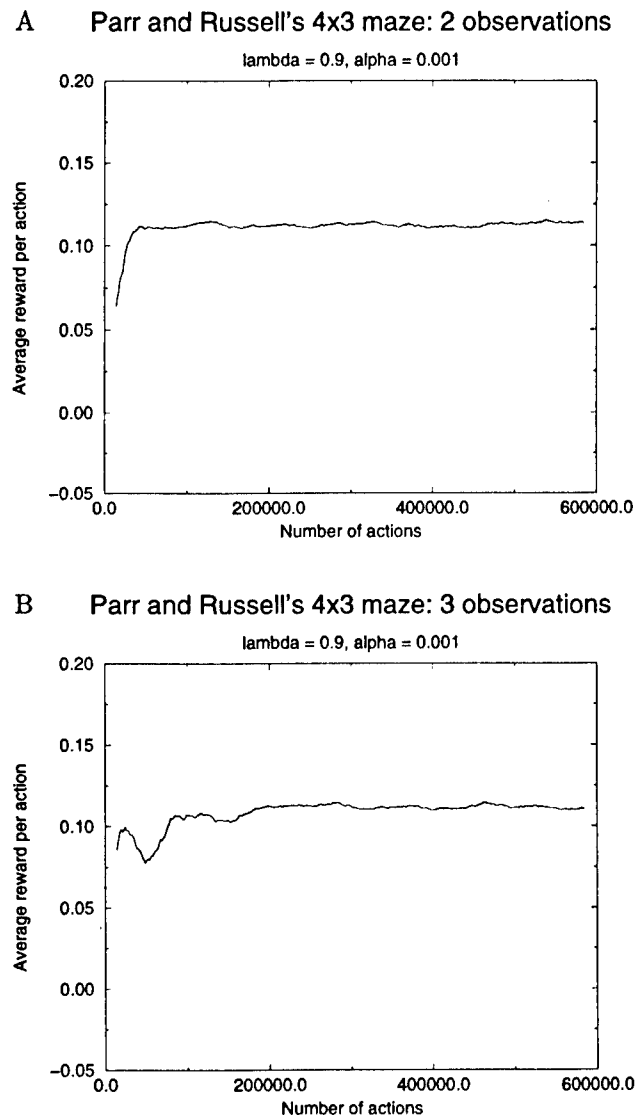


Figure 4: Parr & Russell's Grid World. A) We add one past observation to the immediate observation. The performance of the greedy policy. B) We add two past observations to the immediate observation. The performance of the greedy policy. Note the different scales.

fied that Sarsa(0.9) found the optimal memoryless policy by evaluating the performance of all 4096 possible policies. In this problem, the best memoryless policy is rather poor compared to policies which use memory. The best memoryless policy yields an average reward per step of 0.024 compared to the memory-based policy found by the Witness algorithm (Littman et al., 1995) which yields an average reward per step of 0.1108.

Parr & Russell's SPOVA-RL (Smooth Partially Observable Value Approximation Reinforcement Learning) algorithm learns a value function over belief states and did even better yielding an average reward per step of 0.12 with a memory-based policy¹.

The poor relative performance of the optimal memoryless policy is due to the non-optimal actions the agent must take in the aliased states. For example observation 0 (see Figure 3A) is observed for 3 states in the grid. The state to the left of the penalty state is observed as observation 0 and causes the optimal action in observation 0 to be move north instead of move east. This causes the agent to continuously bump into the upper left corner wall until the transition noise causes a transition to the state to the east.

We investigated the performance improvement obtained by Sarsa(λ) when the immediate observation is augmented with 1 and with 2 previous observations. The performance of the policy using 1 previous observation yielded an average reward per step of 0.1124 (see Figure 4A) which is better than the policy found by the Witness algorithm and almost as good as the policy found by SPOVA-RL. Sarsa(λ) required fewer than 60 CPU seconds to find its policy compared to the 42 CPU minutes for SPOVA-RL and the 12 CPU hours required by the Witness algorithm (Parr & Russell, 1995). The 3-observation performance is shown in Figure 4B and is the same as the 2-observation performance.

We were able to verify that the policy found by Sarsa(λ) using 1 previous observation was indeed the optimal policy in that space. Only ten 2-observation sequences are encountered in the gridworld leading to $4^{10} = 1,048,576$ possible 2 observation policies. We evaluated the performance of all possible 2-observation policies and again verified that the policy found by Sarsa(λ) was the same as the best 2-observation pol-

icy.

4.4 Chrisman's Shuttle Problem

States: Chrisman's (1992) shuttle problem involves an agent operating in an environment with 8 states, 3 actions, and 5 observations. The scenario consists of two space stations with loading docks. The task is to transport supplies between the two docks. There is noise in both the state transitions and observations.

Actions: The agent can execute one of 3 actions: go forward, backup, and turn around.

Observations: The 5 observations are: can see the least recently visited (LRV) station; can see the most recently visited (MRV) station; can see that we are docked in most recently visited (MRV) station; can see that we are docked in least recently visited (LRV) station; and can see nothing. There is sensor noise causing the agent to make faulty observations.

Rewards: The agent receives a reward of +10 when it docks with the least recently visited station. The agent must back into the dock to dock with the station. If the agent collides with the station by moving forward it receives a reward of -3. All other action rewards are 0.

4.4.1 Sarsa(λ) Results

Every 1000 steps (actions) the performance of the greedy policy is evaluated. The performance metric is the average reward per step for 101 trials of up to 101 steps (actions) each. There are (3 actions, 5 observations) $3^5 = 243$ possible memoryless policies. Sarsa(0.9) finds a memoryless policy which yields an average reward per step of 1.02 (see Figure 5A for the learning curve). We verified that the policy found by Sarsa(0.9) was indeed the optimal memoryless policy by evaluating the performance of the 243 possible memoryless policies.

The two best memory-based policies for Chrisman's shuttle problem found by Littman et al. (1995) were found through truncated exact value iteration and their Qmdp method. Truncated exact value iteration found a policy with an average reward per step of 1.805 while Qmdp yielded 1.809. The performance of the optimal memoryless policy is rather poor compared to the performance of policies using memory. This is due to the conservative nature of the optimal memoryless policy which avoids any forward actions so as to avoid receiving the -3 penalty for hitting the station while moving forward.

¹Parr and Russell state that their implementation of the Witness algorithm did not converge on this problem, which probably accounts for the better performance of SPOVA-RL relative to the exact Witness algorithm.

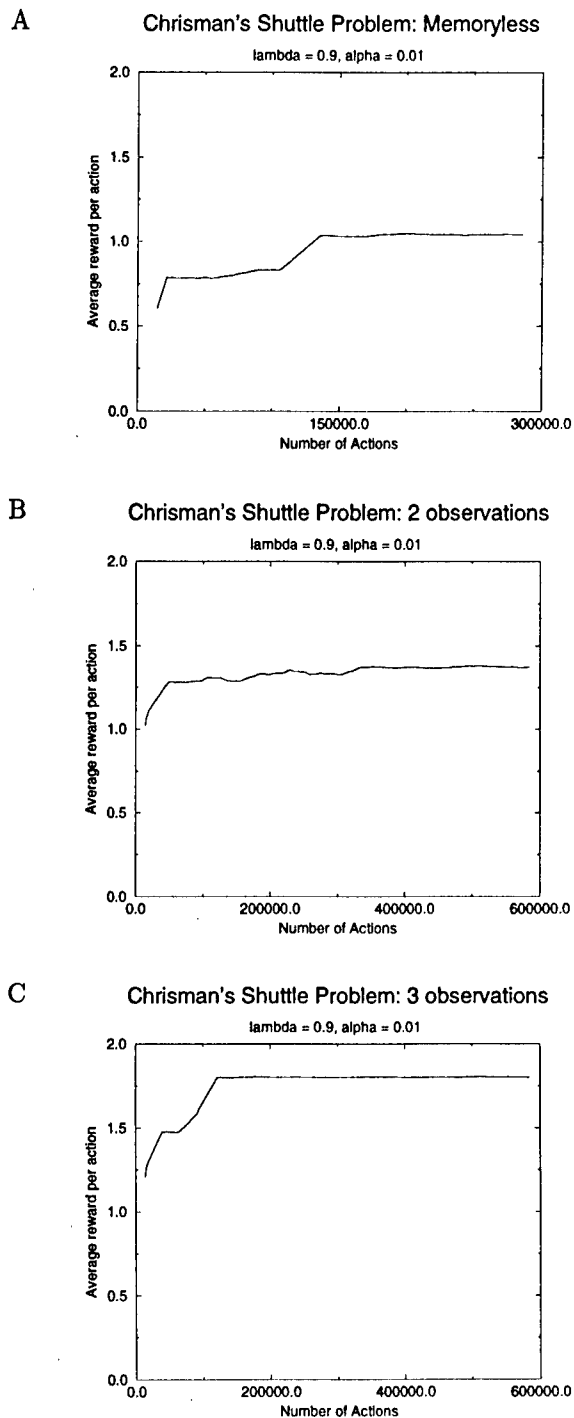


Figure 5: Chrisman's shuttle problem. A) The average reward per action of the memoryless greedy policy as a function of the number of learning steps. B) We add one past observation to the immediate observation. The performance of the greedy policy. C) We add two past observations to the immediate observation. The performance of the greedy policy.

We also investigated the performance improvement obtained by augmenting the current observation with 1 and 2 previous observations. By including the previous observation the performance improved by 37% to an average reward per step of 1.37 (see Figure 5B). By including the 2 previous observations the performance improved by 80% to an average reward per step of 1.804 (see Figure 5C). The performance of the best policy found by Sarsa with 2 previous observations is as good as the truncated exact value iteration method and the Qmdp method, again at a much lower computational cost.

4.5 Discussion

In all the empirical results presented above either we were able to confirm by enumeration that Sarsa found the best policy representable as a mapping from estimated-states (immediate, or immediate and past 1 or past 2 observations) to actions, or in cases where it was not possible to enumerate we observed that Sarsa did as well as the algorithms presented by the originators of the specific POMDPs. Speculating from these empirical results, we conjecture that Sarsa(λ) may be hard to beat in problems where there exists a good policy that maps the observation space to actions.

4.5.1 Why do Eligibility Traces Work?

Consider the set of states that map onto the same observation x . The neighbours of this set of states for some action a may map to several different observations. This can lead to conflicting pulls for the Q-value of x, a depending on which state is providing the experience; some may suggest a is good, some may suggest that a is bad. However these different pulls could get resolved if we considered what happens after n steps. Indeed if we wait until we get to the goal (Monte-Carlo or Sarsa(1)) there would be no confusion due to the hidden state at all. Eligibility traces allow an observation-action pair to access what happens many time steps later, bridging the gap to unambiguous information about the quality of an action. This reasoning indicates that there may be a minimum problem-specific λ that would be needed to bridge the smallest such "gap" in each problem. Our observations during the current work support this; however a careful analysis remains as future work.

5 Conclusion

Partial observability is inevitable in many sequential decision problems of interest to both AI and engineer-

ing. Given the worst-case computational intractability of POMDPs, it is useful to identify sub-classes of POMDPs and algorithms that work well in them. We believe that eligibility trace based RL methods such as Sarsa(λ) can be useful in POMDPs that have good memoryless or good low-order-memory-based policies. We demonstrated this empirically on four POMDP problems from the recent literature. A more powerful result that remains future work would be to prove this theoretically.

Acknowledgements

Satinder Singh was supported by NSF grant IIS-9711753. We thank Michael Littman and the anonymous reviewers for many valuable comments.

References

- Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 835–846.
- Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI-92*.
- Jaakkola, T., Singh, S., & Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352. Morgan Kaufmann.
- Lin, L. J. & Mitchell, T. M. (1992). Reinforcement learning with hidden states. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*.
- Littman, M., Cassandra, A., & Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA. Morgan Kaufmann.
- Littman, M. L. (1994). Memoryless policies: theoretical limitations and practical results. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*.
- McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. In Utgoff, P. (Ed.), *Machine Learning: Proceedings of the Tenth International Conference*, pages 190–196. Morgan Kaufmann.
- Parr, R. & Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Rummery, G. A. & Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Dept.
- Singh, S., Jaakkola, T., & Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In Cohen, W. W. & Hirsh, H. (Eds.), *Machine Learning: Proceedings of the Eleventh International Conference*, pages 284–292. Morgan Kaufmann.
- Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: discounted case. *Operations Research*, 26, 282–304.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 9–44.
- Sutton, R. S. (1990). Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA. Morgan Kaufmann.
- Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.
- Tesauro, G. J. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3), 58–68.
- Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge Univ., Cambridge, England.
- Whitehead, S. D. & Ballard, D. H. (1990). Active perception and reinforcement learning. In *Proc. of the Seventh International Conference on Machine Learning*, Austin, TX. M.

Learning To Locate An Object in 3D Space From A Sequence Of Camera Images

Dimitris Margaritis

Dept. of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
dmarg+@cs.cmu.edu

Sebastian Thrun

Dept. of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
thrun+@cs.cmu.edu

Abstract

This paper addresses the problem of determining an object's 3D location from a sequence of camera images recorded by a mobile robot. The approach presented here allows people to "train" robots to recognize specific objects, by presenting it examples of the object to be recognized. A decision tree method is used to learn significant features of the target object from individual camera images. Individual estimates are integrated over time using Bayes rule, into a probabilistic 3D model of the robot's environment. Experimental results illustrate that the method enables a mobile robot to robustly estimate the 3D location of objects from multiple camera images.

1 INTRODUCTION

In recent years, there has been significant progress in the field of mobile robotics. Applications such as robots that guide blind or mentally handicapped people, robots that clean large office buildings and department stores, robots that assist people in recreational activities, etc., are slowly getting in reach. Many of these robots must integrate mobility with manipulation. They must be able to move around, and they must also be capable of manipulating their environment. For such robots, their practical success will partially depend on their ability to identify and localize objects.

This paper addresses the problem building robots that can be *trained* to recognize and locate user-specified objects. More specifically, it proposes an algorithm that enables people to train robots by simply showing a few poses of the object. Once trained, the robot can recognize these objects and determine their location in 3D space. In contrast to

existing approaches to mobile manipulation, which usually assumes that objects are located in floor or table-height, our approach does not make restrictive assumptions as to where the object is located. This poses new challenges on the ability to localize objects, as a single camera image is insufficient to determine the location of an object in 3D space.

The approach proposed here uses probabilistic representations to estimate the identity and location of the target object from multiple views. It maps camera images into 2D probabilistic maps, which describe, for each pixel in the camera image, the likelihood that this pixel is part of the target object. This mapping is established by a decision tree applied to local image features, which is constructed during the training phase from labeled images. The 2D probabilistic map is then projected into the 3D work space, based on straightforward geometric considerations. Since a single camera image is insufficient to determine the location of an object in 3D, our approach integrates information from multiple images, taken from multiple viewpoints. It employs Bayes rule to generate a consistent probabilistic 3D model of the workspace. Our approach also takes into account the uncertainty introduced by robot motion, by using a probabilistic model of robot motion. As the robot moves in the environment taking images, it gradually improves the estimation of the identity and location of an object, until it finally knows what and where the object is. Experimental results using a RWI B21 robot equipped with a color camera show that multi-part objects can be located robustly and with high accuracy.

The remainder of this paper is organized as follows. In section 2, we briefly describe decision trees along with the way our approach uses them for characterizing images. In section 3, we show how image information is integrated into a 3D model, and provide a method for accommodating the uncertainty that is introduced by robot motion. In section 4, we present experimental results, obtained with a RWI B21

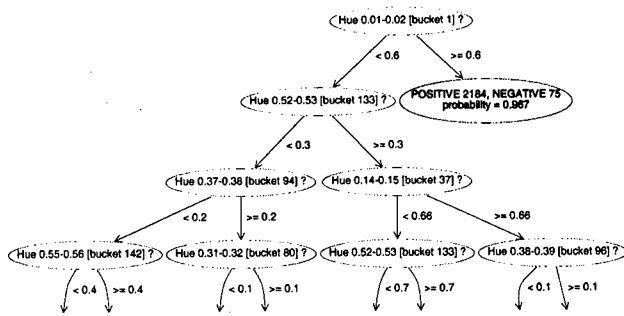


Figure 1: The top few nodes of an example decision tree. A leaf represents the probability conditioned on the values of the attributes. Internal nodes test on the fraction of positive pixels of a tile that fall in the corresponding hue range.

robot, followed by a survey of related research (section 5). Finally, in section 6, we comment on the assumptions and limitations of the approach and suggest directions for future research.

2 DECISION TREE LEARNING

A decision tree is a succinct and explicit way of representing a multidimensional discrete-valued function $f : \mathcal{R}^n \times \mathcal{X}^m \rightarrow \mathcal{Y}$, where \mathcal{X} and \mathcal{Y} are finite sets of discrete elements and \mathcal{R} is the set of real numbers. The $(n + m)$ inputs to this function frequently correspond to discrete and/or continuous-valued attributes of an object and the output represents an object's property that we want to predict. Each node of the tree is associated with a partition of the input space. An internal node further partitions its space into two subspaces based on the value of a single input variable, associating each of the resultant subspaces to each of the two children. The set of decision trees is complete in the space of discrete-valued functions i.e. any such function can be represented by at least one decision tree. An example of a decision tree, obtained in the context of image analysis in a fashion similar to the one used in this paper (see below), is illustrated in Fig. 1.

Our approach uses decision trees to approximate conditional probability density functions. Decision trees are usually used to answer YES/NO queries regarding the output value of f given an input tuple of values. If, for example, f is a boolean-output function, querying is typically done by comparing the number of positive and the number of negative training examples that were assigned during training to the leaf node that is associated with the partition that the input tuple lies in. The algorithm would then return the value (YES/NO) that is in majority in that leaf. In our use of a decision tree we differ in that we instead output the fraction

of positive or negative examples found in the leaf. As such, we use the decision tree to represent an *approximation of the probability density function* on the output space conditioned on the values of attributes in the input space of f . If appropriately pruned (during a post-pruning phase that is intended to increase compactness and, more importantly, generalization over future data), these probabilities are usually not zero or one because of training set noise in either the values of the inputs or the output or non-determinism due to use of a set of input variables that is insufficient to deterministically model f .

2.1 A PDF FOR CHARACTERIZING AN IMAGE

Our approach uses a decision tree to map (filter) camera images into 2D probabilistic maps, which describe the probability of the presence of a target object at the various locations in the image. More specifically, the inputs to the tree are *image features* in a local region (called: *tile*) in the image, and the output is a probability value that measures the likelihood of the presence of a target object in the respective tile. In principle, our approach can be applied to arbitrary image features (e.g., pixels, edges, brightness, color, texture, etc.). In our implementation, *local color histograms* are used as input to the decision tree.

The tree is learned using labeled training examples. More specifically, construction of the training, test and pruning sets is done using the following procedure:

1. An input picture is obtained.
2. A rectangle R is drawn around the object by the user. This might include parts of the background.
3. The image is divided in a matrix of non-overlapping rectangular tiles, completely covering its surface. The size of each tile is small relative to the projection of the object on the image. 8×8 is used in this paper.
4. Each tile is used to construct a single positive or negative example. The features that occur in the tile, which can be continuous or discrete, are extracted and used as input values for the example associated with that tile.
5. Depending on whether each tile is fully contained within R or not, the example is assigned to be positive or negative, respectively.

This set of examples is equally divided into training, test and pruning sets, and these are used in growing a decision tree for that combination of object and environment that it was seen in. The resulting tree, when applied to new images within that environment, provides probability densities for the presence of a target object.

Figure 2 illustrates our method. Shown there, in the top

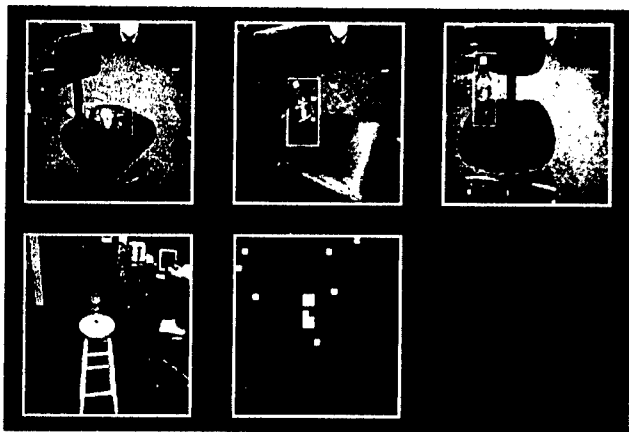


Figure 2: Detection of a bottle from previous examples. The top row contains images where the outlined part contains the tiles used as positive examples. The rest of the image's tiles are negative examples. Probabilities above 0.8 are marked in the previously unseen picture in the bottom row. Not shown is another set of 18 "background" pictures consisting of negative examples only.

row, is a series of three training images. The target object is labeled by hand. The bottom row shows a test image, along with the probability field generated by the tree. As can be seen there, the algorithm assigned high likelihood to the correct location, but also misclassified a small number of regions in the image background. From this single camera image, it is impossible to determine the location of the target object in 3D coordinates. The remainder of this paper describes our approach to integrating these probabilistic estimates in 3D space.

3 INTEGRATING MULTIPLE CAMERA IMAGES IN 3D

Our approach integrates the probabilistic information, extracted from individual images, into a spatial 3D model of the world. Information about the location of the object is represented as a 3D occupancy grid. Each grid cell is associated with an approximation of the probability that part of the object occupies that particular cell. Each such probability is initialized with a number that corresponds to a prior belief that the object occupies a cell given no information about the world. This number can be learned from data, typically though counting according to the frequentists' approach to probability. The exact value of the prior is not significant in the long term, since the value will converge towards the actual probability after a sufficient number of observations. However, if there is evidence that the object in question occurs more frequently in certain areas

(for example, a shoe may be expected to lie on the floor most of the time), this information can be used to appropriately initialize prior probabilities and assign higher values to these locations. During detection, each information-gathering step is followed by an updating of the probability of each cell according to Bayes law, as described below. Robot motion also affects the grid due to uncertainty of the robot's translational and rotational velocities.

3.1 INFORMATION INTEGRATION

The key idea for mapping 2D image information into a 3D spatial representation is to map image tiles into *pyramids* in space. Each image obtained from the environment provides us with information about the location of parts of the object. Since we assume a single camera input, we have no information about the depth of features contained in one of the tiles of the image. We therefore make no assumption on the distance of the part from the eyepoint. However, we do obtain information about the Euler angles (azimuth θ and altitude ϕ) of the feature with respect to the robot's current location. In particular we know that it is contained within the pyramid emanating from the eyepoint whose four converging sides intersect the four corners of the tile on the image plane that is perpendicular to the direction the camera is facing. Grid cells intersecting this pyramid are therefore updated using Bayes law.

An example of the updating is shown in Fig. 3. Here two different pyramids are shown (projected into the x - y plane), which have been generated from camera images taken at different locations. Bayes rule is applied to integrate these pyramids, in order to generate a single, consistent belief.

The integration works as follows. The probability that a part of the object occupying a cell at grid location (x, y, z) at time t is denoted by $\Pr[\xi(x, y, z, t)]$. Coordinates x, y and z are with respect to a fixed, world-centered coordinate system (they are not local robot-centered coordinates). $\xi(x, y, z, t)$ is a boolean random variable denoting the existence of a part of the object at a location somewhere inside the corresponding grid cell. In the following we will use ξ instead of $\xi(x, y, z, t)$ for the sake of brevity. If $i(t)$ denotes the image obtained at time t and $D(t-1)$ the set of previous images/motion commands in all previous steps, the probability value $p(\xi)$ at grid cell location ξ is computed as follows:

$$\begin{aligned} p(\xi) &\equiv \Pr[\xi \mid i(t), D(t-1)] \\ &= \Pr[i(t) \mid \xi, D(t-1)] \frac{\Pr[\xi \mid D(t-1)]}{\Pr[i(t) \mid D(t-1)]}. \end{aligned}$$

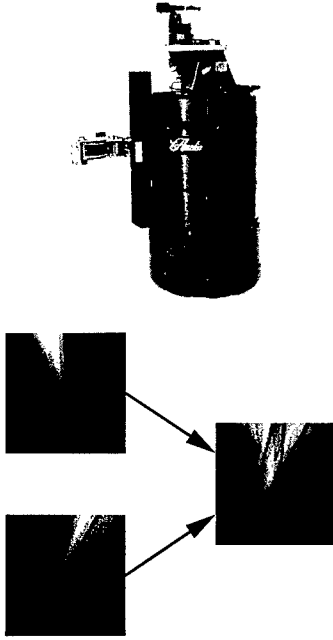


Figure 3: On the top, the robot used in our experiments is shown. It is equipped with a parallel two-fingered gripper for object manipulation. On the bottom, an illustration is presented of how information from images taken from two different viewpoint is integrated in the occupancy grid. Shown to the left are two single projections applied to an "empty" grid. The picture on the right shows how they are combined together. The images depict the average values of grid cell probabilities when viewed from above (i.e. averaging probability values along the z -axis).

$\Pr[\xi \mid D(t-1)]$ is the prior probability accumulated in the cell from previous iterations of the procedure, which takes into account all previous data. $\Pr[i(t) \mid \xi, D(t-1)] = \Pr[i(t) \mid \xi]$ by making a Markov conditional independence assumption that implies that, given the *fact* of the existence or not of part of the object in the cell, the image obtained does not depend on previous images. Under this assumption, by using

$$\Pr[i(t) \mid \xi] = \frac{\Pr[\xi \mid i(t)] \Pr[i(t)]}{\Pr[\xi]}$$

we obtain

$$p(\xi) = \frac{\Pr[\xi \mid i(t)] \Pr[i(t)]}{\Pr[\xi]} \frac{\Pr[\xi \mid D(t-1)]}{\Pr[i(t) \mid D(t-1)]}$$

and

$$p(\bar{\xi}) = \frac{(1 - \Pr[\xi \mid i(t)]) \Pr[i(t)]}{1 - \Pr[\xi]} \frac{1 - \Pr[\xi \mid D(t-1)]}{1 - \Pr[i(t) \mid D(t-1)]}$$

where $\bar{\xi}$ is the complement of event ξ . $\Pr[\xi \mid i(t)]$ is the probability estimate returned by the decision tree for the

tile corresponding to the cell at (x, y, z) by only taking the current image into account. In estimation problems of this type, it is common practice to compute the *odds-ratio*, for which $\Pr[i(t)]$ and $\Pr[i(t) \mid D(t-1)]$ cancel out:

$$\begin{aligned} \text{odds-ratio}(\xi) &\equiv \frac{p(\xi)}{p(\bar{\xi})} \\ &= \frac{\Pr[\xi \mid i(t)]}{1 - \Pr[\xi \mid i(t)]} \frac{\Pr[\xi \mid D(t-1)]}{1 - \Pr[\xi \mid D(t-1)]} \times \\ &\quad \frac{1 - \Pr[\xi]}{\Pr[\xi]} \Rightarrow \\ p(\xi) &= \frac{\text{odds-ratio}(\xi)}{1 + \text{odds-ratio}(\xi)}. \end{aligned}$$

Similar formulas for belief integration can be found in [Pea88, Thr98b].

3.2 ROBOT MOTION

Each robot motion introduces uncertainty into the robot's estimate of the object's location because of imperfect actuators and measuring devices. We model the translational as well as rotational magnitude of the velocity of the robot as a Gaussian random variable with mean equal to the nominal velocity given to the robotic motion controller—we make the assumption that there are no systematic errors. The standard deviations used are pessimistic estimates of the deviation around the nominal corresponding velocity magnitude. The accurate determination of the standard deviations does not significantly influence our location estimates given frequent enough observations. Under this assumption, their actual value is not critical and can be overestimated.

If the magnitude of the velocity is normally distributed with mean v_0 and standard deviation σ_v , $v \sim N(v_0, \sigma_v^2)$ (assume one-dimensional for the purpose of this example), the location of a object with that velocity after time t is a random variable $x \sim N(v_0 t, \sigma_v^2 t^2)$, also normally distributed, with mean $v_0 t$ and standard deviation $\sigma_v t$. This suggests that uncertainty of an objects location increases with time as time goes by, as shown in Fig. 4.

4 EXPERIMENTAL RESULTS

We conducted our experiments on a B21 mobile robot equipped with a single Sony XC-999 color camera with a 6mm focal length lens, mounted on a pan-tilt unit. Images of size 240×256 are acquired through a Matrox Meteor

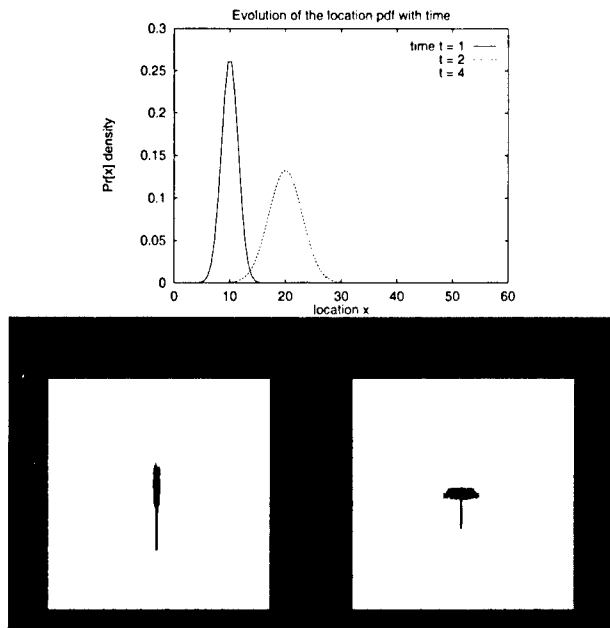


Figure 4: Probabilistic model of robot motion. Top image: Belief of the location of the object deteriorates in time under uncertainty of the magnitude of the velocity. Here $v \sim N(10, 1^2)$. Bottom image: This graph illustrates the outcome of specific motion commands projected along the z axis (a translation and a rotation).

framegrabber connected to the camera and are used to train a decision tree in the manner described in section 2.1.

We chose a simple histogram representation of downsampled versions of the training images as the input features to our decision tree algorithm. In particular, we use color histograms for each tile, at resolution of 256 color bins. Therefore each tile represents an example of 256 input features, namely the pixel percentages at each color bin, and one binary-valued output, corresponding to the event that the tile is part of the object being trained on. Even though this choice of input features does not take into account all information present in the picture, this is simply an artifact of the current implementation and by no means imposes any restriction on the choice of input features of the approach in general. More complex features may be employed in future implementations. However, as we demonstrate below, this simple representation performs adequately well in certain frequently occurring situations where the object is sufficiently distinct from the background, containing enough information for recovering the approximate location of simple objects in 3D. The “distinctiveness” is determined by the resolution of our color histogram, coupled with the amount of hue variation that changes in light intensity on the object result in.

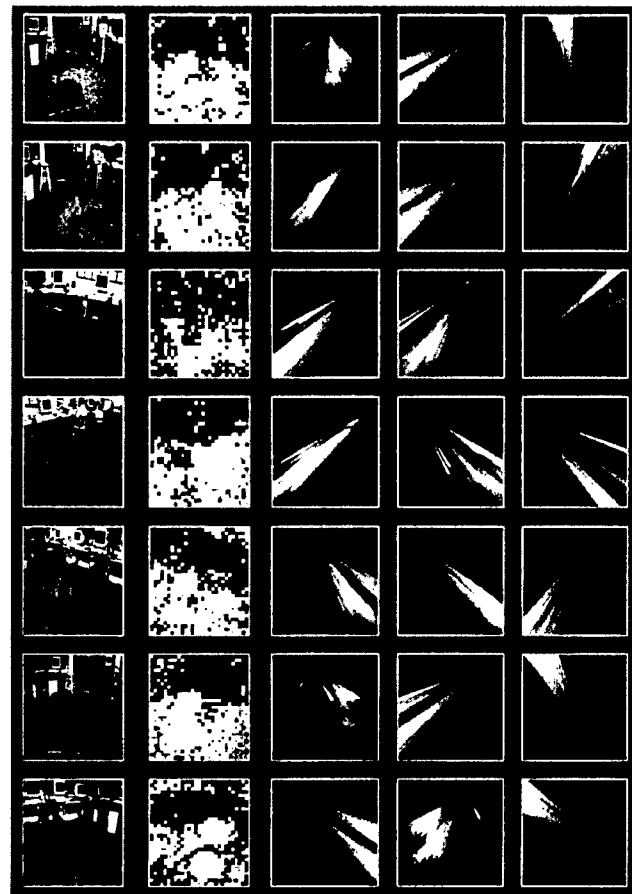


Figure 5: Probability map that is the output of the decision tree trained to recognize the red chair. The brightest tiles in the probability map (second column) correspond to probability greater than 0.9. Projection of the map in 3D are shown in the last three columns, as averages along the x , y and z (rightmost column) axis respectively.

An example application of a decision tree trained on three examples with an object (in this case, a bottle) and 18 background images (containing negative examples only) is shown in Fig. 2. The top few nodes of the tree are shown in Fig. 1. In a similar fashion we constructed a decision tree to recognize a larger simple object, a red chair, by using the same all-negative example images and three additional images containing the chair in different poses. We then manually maneuvered the mobile robot around the chair taking 7 new pictures from different angles. These pictures are shown in Fig. 5. The second column in that figure depicts the probability map that is output from the decision tree for each image. At certain locations we acquired images and projected the probability map in 3D, with each probability map element corresponding to a pyramid, as described in 3.1. Every cell covered by a pyramid is affected

by the corresponding probability in the probability map. The results of projection when viewed along the x , y and z axes are shown in the three rightmost columns in Fig. 5. Each pixel in these projections has intensity proportional to the average probability along the axis of projection passing from that pixel. The z -axis projections make the locations around the chair that the pictures were taken particularly easy to see.

In reality, the robot does not keep a 3D grid for each image but rather incorporates information incrementally in the single grid it maintains, which is justified under the Markov assumption. This is done by applying Bayes law for each cell individually. There is no normalization done over the whole grid, which corresponds to the semantics we assign to the probability stored at each cell: it represent the probability that a part of the object occupies that cell. As such, we make no assumptions about the size of the object with respect to the cell size.

Between images, the robot is maneuvered manually to the spot where the next image will be taken. These motions increase our uncertainty in the manner described in section 3.2. The robot used in the experiments is a semi-holonomic one, its motion consisting of rotations and forward or backward motions in the direction it is facing. As such we model rotational and translational uncertainty in the magnitude of the velocity.

The updating of the grid using the above procedure is shown in Fig. 6 for one run. This sequence of beliefs corresponds to a situation where a robot faces a chair. The grid size used is $100 \times 100 \times 100$ and each unit along any direction corresponds to 4cm in the real world. All beliefs shown in Fig. 6 are projected horizontally.

As can be seen in Fig. 6, the initial location of the target object(s) is unknown. After taking a first image, the robot's belief is a conjunction of pyramids, corresponding to the output of the decision tree. As the robot moves, it loses information. As it takes the second snapshot from a different perspective, the belief is refined. After taking seven images, the location and the shape of the target object are reconstructed with high accuracy. As these results demonstrate, our approach can accurately determine the location of the target object. It is also robust to errors in the robot's odometry. This robustness is a result of incorporating our probabilistic model of robot motion.

5 RELATED WORK

Decision trees [Qui86, Qui93, Mit97] are one of the most popular inductive machine learning method to date. The early algorithms were only applicable to problems with

discrete input and output spaces. Decision tree learning algorithms in AI for real-valued input spaces were proposed by [BFOS84], as a reinvention of earlier work. Tree-based regression methods for real-valued input and output spaces can also be found in [Fri91, Moo90]. The work presented in this paper provides an example where a decision tree is used to learn a conditional probability density function. Like the approaches presented in [FI93, MKS94], it partitions a real-valued high-dimensional input space into hypercubes. The output nodes, however, represent conditional densities, which are estimated using a frequentist approach [CB90]. This is related to results reported in [TLS89, Mac92, Mit97], which show that under appropriate assumptions, artificial neural networks approximate conditional probability density functions.

The mathematical approach for integrating information is adopted from the statistical literature [CB90, Pea88]. The approach presented in this paper also bears close resemblance to occupancy grids [Mor88, Elf89]. Occupancy grid approaches are popular techniques for learning models of mobile robot environments from sensor data. Just like the approach proposed here, they represent the environment using fine-grained, evenly spaced grids. Each grid point is annotated by a probability, which describes the evidence that a location contains an object/obstacle. The vast majority of existing approaches differs from the one proposed here in three aspects. First, they model occupancy, not the location of a specific target object. Second they are usually constructed from range measurements (e.g., sonar, laser), not from camera images. Third, they are usually two-dimensional. There are, however, notable exceptions. The approaches described in [MM94, TBB⁺98] construct occupancy grids from sequences of camera images. Moravec and Martin's approach [MM94] has probably been the first to construct 3D grids, instead of the commonly used 2D representations. Both approaches, however, used stereo vision to estimate the location of obstacles. Stereo vision generates distance estimates, which greatly facilitates the construction of the maps. The approach reported here estimates distance indirectly, through integrating multiple camera images recorded at different locations. Unfortunately, the approach in [MM94] is incapable of dealing with error in the robot's odometry.

Object-centered 3D object reconstruction has also been investigated in the context of computer vision. Two approaches have emerged. One models objects as 3D surfaces, typically represented as a polygonal meshes. For example, [FL95] uses stereo and intensity matching to construct and fit the mesh. The second approach uses a grid representation essentially similar to the one used in this paper (e.g. [Col96]), and employs a technique sometimes re-

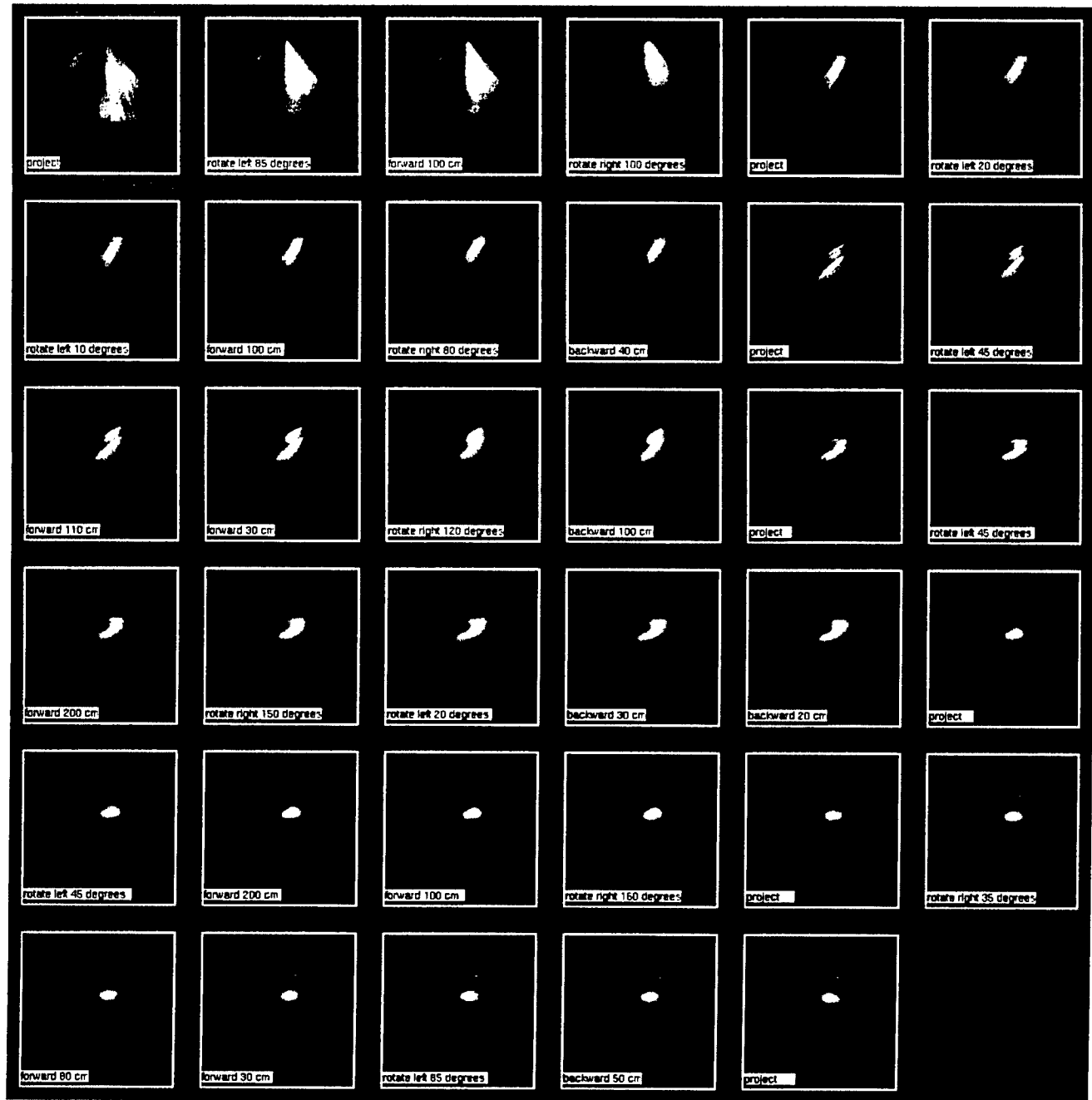


Figure 6: Cumulative effects on motion and probability map projection on grid as viewed along the x -axis (that is running perpendicular from the door facing the interior of the room in the pictures in Fig. 5). The two distinct parts of the chair (back and seat) are discernible.

ferred to as "3D voting" to update cell "occupancies." This differs from our approach in two ways: first, cells are updated by counting votes in a straightforward if ad hoc manner which employs techniques such as voting for cells in a radius of the intersecting with the line through the eyepoint and the line segment. This is necessitated partly from the inability to model inaccuracies in the viewpoint location, although in many such applications—for example, military aerial photography—the camera location is estimated relatively accurately. Second, these techniques do not learn a probabilistic model of the set of features that are employed from examples. As such, all features are equally weighted, necessitating the use of a threshold—in order to produce a recognizable picture—the selection of which can be difficult (although see [Col96] for a statistical approach to threshold estimation).

Our approach is similar to Markov localization [BFHS96, NPB95, SK95, Thr98a], a method for probabilistically estimating the pose of a mobile robot in a (known) environment. Markov localization relies on the same statistical principles for integrating multiple sensor readings into a single belief. In fact, the approach in [BFHS96] uses the same basic representations as our approach: evenly spaced grids. Markov localization, however, rests on the assumption that there is exactly one object (*i.e.*, the robot) whose location is to be estimated. Our approach can handle situations that contain a variable (unknown) number of target objects.

Finally, the problem of finding and manipulating objects has received considerable attention within the AI community (see [Hor94] and various papers in [Sim95, KBM98]). For example, Buhmann et al. [BBC⁺95] described an approach where a robot could be trained to recognize specific objects. Most existing approaches in the mobile robot community, however, make the assumption that the object is located in floor-height, in which case camera coordinates can directly be converted to real-world coordinates. Our approach is specifically designed to find objects at arbitrary locations in space. This is important in many real-world applications, as objects may frequently be found in tables, chairs, etc.

6 DISCUSSION AND FUTURE RESEARCH

This paper presented a novel approach to estimating the 3D location of an object with a mobile robot. Individual camera images are interpreted using a decision tree method, which maps image regions (tiles) into probabilistic estimates for the presence of target objects. Based on a straightforward geometric consideration, these probabil-

ities are mapped into 3D pyramids in global world coordinates. Multiple pyramids, obtained from camera images recorded from different viewpoints, are integrated using Bayes rule into a single probabilistic model of the object location. Noise in robot motion is accounted for by a probabilistic model of robot motion. Experimental results demonstrate that the method can robustly localize objects in 3D space.

A key advantage of the current approach is its generality. No assumption is made concerning the typical location of objects (*e.g.*, they are not assumed to lie on the floor). The approach can also be trained easily to recognize new, user-specified objects. While our current implementation uses color as the primary cue for object recognition, the method can equally be applied to a much richer range of image features, making it fit for a large class of target objects (*i.e.*, objects that can be recognized from local image features).

Our approach rests on several limiting assumptions. First of all, it assumes that object does not move. To accommodate moving objects, our approach would have to be extended by a probabilistic model of object motion. Such a model might characterize the *typical* motion speed of the target object. It is unclear, however, if such an approach would be able to gather sufficient information to estimate the location of a moving object with the necessary accuracy.

Our approach also assumes that the training images accurately represent the situation during testing. In our experiments, we usually enriched the training set by a small number of pictures recorded at random locations in our lab. These pictures were used as negative training examples when growing the tree. We found that these additional images increased the robustness of the image analysis, thereby improving the overall estimation results. However, the method might fail if the robot encounters an object which similar to the target object, but which has not been part of its training set.

The spatial resolution in the experiments described in this paper is low, due the enormous complexity involved in updating 3D grids. By choosing a 4cm resolution, the computational overhead was manageable. Denser and larger grids are desirable, but unfortunately the computational cost of updating the grid is cubic in the number of grid cells. An interesting extension of the current approach would be to use variable-resolution representations, such as oct-trees [Sam89b, Sam89a, Moo90], for representing object location. Such representations could balance the computational and memory resources, by modeling regions coarsely that are unlikely to contain a target object. If the density of target objects is low (which is usually the case), such an ex-

tension could improve the computational efficiency of the approach substantially.

Another promising extension of the current approach would be to devise methods that *actively* control the robot so as to maximize information gain. In the experiments presented here, a human manually positioned the robot. In our previous work [Thr98b], however, we already developed successful methods for active information gathering, which were applied in the context of learning 2D occupancy grid maps. In the context of object localization, such methods could lead to a behavior where a robot investigates the object from multiple viewpoints, in order to estimate its location accurately. The development of such methods is subject to future research.

References

- [BBC⁺95] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot Rhino. *AI Magazine*, 16(1), 1995.
- [BFHS96] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, August 1996. AAAI Press/MIT Press.
- [BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Belmont: Wadsworth, 1984.
- [CB90] G.C. Casella and R.L. Berger. *Statistical Inference*. Wadsworth & Brooks, Pacific Grove, CA, 1990.
- [Col96] R. T. Collins. A space-sweep approach to true multi-image matching. *IEEE Computer Vision and Pattern Recognition*, pages 358–363, 1996.
- [Elf89] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.
- [FI93] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of IJCAI-93*, Chamberry, France, July 1993. IJCAI, Inc.
- [FL95] P. Fua and Y. Leclerc. Object-centered surface reconstruction: Combining multi-image stereo and shading. *IJCV*, 16(1):35–56, 1995.
- [Fri91] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, March 1991.
- [Hor94] I. Horswill. Specialization of perceptual processes. Technical Report AI TR-1511, MIT, AI Lab, Cambridge, MA, September 1994.
- [KBM98] D. Kortenkamp, R.P. Bonassi, and R. Murphy, editors. *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA, 1998. MIT Press. to appear.
- [Mac92] D. J. C. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, Pasadena, California, 1992.
- [Mit97] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [MKS94] S.K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–33, 1994.
- [MM94] H.P. Moravec and M.C. Martin. Robot navigation by 3D spatial evidence grids. Mobile Robot Laboratory, Robotics Institute, Carnegie Mellon University, 1994.
- [Moo90] A. W. Moore. *Efficient Memory-based Learning for Robot Control*. PhD thesis, Trinity Hall, University of Cambridge, England, 1990.
- [Mor88] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.
- [NPB95] I. Nourbakhsh, R. Powers, and S. Birchfield. DERSH an office-navigating robot. *AI Magazine*, 16(2):53–60, Summer 1995.
- [Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.
- [Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
- [Sam89a] H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley Publishing Inc., 1989.
- [Sam89b] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Inc., 1989.
- [Sim95] R. Simmons. The 1994 AAAI robot competition and exhibition. *AI Magazine*, 16(1), Spring 1995.
- [SK95] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of IJCAI-95*, pages 1080–1087, Montreal, Canada, August 1995. IJCAI, Inc.
- [TBB⁺98] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Frölinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998. to appear.
- [Thr98a] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 1998. to appear.
- [Thr98b] S. Thrun. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 1998. to appear.
- [TLS89] N. Thishby, E. Levin, and S. A. Solla. Consistent inference of probabilities in layered networks: predictions and generalizations. In *Proceedings of the First International Joint Conference on Neural Networks*, Washington, DC, San Diego, 1989. IEEE TAB Neural Network Committee.

Multiple-Instance Learning for Natural Scene Classification

Oded Maron
Artificial Intelligence Lab
NE43-755, M.I.T.
Cambridge, MA 02139
oded@ai.mit.edu

Aparna Lakshmi Ratan
Artificial Intelligence Lab
NE43-739, M.I.T.
Cambridge, MA 02139
aparna@ai.mit.edu

Abstract

Multiple-Instance learning is a way of modeling ambiguity in supervised learning examples. Each example is a bag of instances, but only the bag is labeled - not the individual instances. A bag is labeled negative if all the instances are negative, and positive if at least one of the instances is positive. We apply the Multiple-Instance learning framework to the problem of learning how to classify natural images. Images are inherently ambiguous since they can represent many different things. A user labels an image as positive if the image somehow contains the concept. Each image is a bag, and the instances are various sub-regions in the image. From a small collection of positive and negative examples, we can learn the concept and then use it to retrieve images that contain the concept from a large database. We show that the Diverse Density algorithm performs well in this task, that simple hypothesis classes are sufficient to classify natural images, and that user interaction helps to improve performance.

1 INTRODUCTION

Scene classification is an open problem in machine vision and has applications in image and video database indexing. We investigate a method for learning visual concepts that encode the properties of a scene class from a small set of positive and negative examples. Extracted concepts are simple templates that capture some color and spatial properties of the class. Work by Lipson [Lipson *et al.*, 1997] illustrates that sim-

ple, hand-crafted templates that describe the relative color and spatial properties in an image can be used successfully to classify natural scenes like fields, snowy mountains and waterfalls. In this paper we show that these templates can be learned. We describe a framework for learning scene-class concepts that can be used effectively for the task of content-based image retrieval from large databases. The learning framework we use in this paper is called Multiple-Instance learning [Dietterich *et al.*, 1997], [Maron and Lozano-Pérez, 1998]. In this framework, examples are not labeled examples, but are labeled bags. Each bag is a collection of instances (Figure 1). A bag is labeled negative if all the instances in it are negative, and positive if at least one of the instances in it is positive. We use this framework to model the ambiguity in mapping an image to many possible templates which describe the image. Specifically, every image is a bag, and each possible template for describing the image is one instance in the bag. We discuss a method called Diverse Density [Maron and Lozano-Pérez, 1998] for learning concepts from Multiple-Instance examples.

We test our approach on images from the COREL photo library. We show that the system is successful even when the hypothesis class involves very simple templates, and even when the images are sampled very coarsely. In addition, we show that user interaction (refining the hypothesis through the addition of more examples) is helpful in improving the performance of the learning system. In Section 2, we discuss previous and related work in image classification. We then describe the Multiple-Instance learning framework and the Diverse Density algorithm. In section 4 we detail our experimental setup and show results on various concept classes, hypothesis classes, and training regimes.

The third contribution of this paper (in addition to

a novel application of Multiple-Instance learning and the discovery that surprisingly simple concepts do well on this task) is the development of a general architecture to combine ideas from the vision and machine learning communities. A key part of our system is the bag generator: a mechanism which takes an image and generates a set of instances, where each instance is a possible description of what the image is about. If an idealized object recognizer existed, then the bag generator would simply output a list of the objects in the image. The learning algorithm would be straightforward: find an intersection between the positive lists that didn't include elements from the negative lists. On the other extreme, if we had a learning algorithm that could handle billions of instances per bag, then we would not need an object recognizer. Instead, the bag generator would simply output every subcombination of pixels in the image. In this paper, we use a slightly more sophisticated bag generator (one that generates subregions), which limits the number of instances per bag and therefore allows us to use an algorithm such as Diverse Density. The key observation is that a better bag generator (progress in the vision community) leads to a simpler learning algorithm, while at the same time a better Multiple-Instance learning algorithm (progress in the machine learning community) allows us to use simpler segmentation algorithms. This is in contrast with the architecture of [Keeler *et al.*, 1991], for example, where the learning mechanism is woven into the position-invariant representation of subimages.

2 IMAGE CLASSIFICATION SYSTEMS

In the past few years, the growing number of digital image and video libraries has led to the need for flexible, automated content-based image retrieval systems which can efficiently retrieve images from a database that are similar to a user's query. Because what a user wants can vary greatly, we also want to provide a way for the user to explore and refine the query by letting the system bring up examples.

One of the most popular global techniques for indexing is color-histogramming which measures the overall distribution of colors in the image. While histograms are useful because they are relatively insensitive to position and orientation changes, they do not capture the spatial relationships of color regions and thus have limited discriminating power. Many of the existing image-querying systems work on entire im-

ages or in user-specified regions by using distribution of color, texture and structural properties. The QBIC system [Flickner *et al.*, 1995] is an example of such a system. Some recent systems that try to incorporate some spatial information into their color feature sets include [Smith and Chang, 1996, Huang *et al.*, 1997, Belongie *et al.*, 1998]. Promising work by Rubner [Rubner *et al.*, 1998] on the earth mover's distance provides a metric that overcomes the binning problems of existing definitions of distribution distances for indexing. Most of these techniques require the user to specify the salient regions in the query image. One of the goals of our system is to learn the relevant color and spatial properties that best describe a particular class of natural scenes.

More recently, work by Lipson and Sinha ([Lipson *et al.*, 1997]) in scene classification illustrates that pre-defined flexible templates that describe the relative color and spatial properties in the image can be used effectively for this task. The flexible templates constructed by Lipson [Lipson *et al.*, 1997] encode the scene classes as a set of image patches and qualitative relationships between those patches. Each image patch has properties in the color and luminance channels. These templates describe the color relationship (relative changes in the R,G,B channels), luminance relationship (relative changes in the luminance channel) and spatial relationship between two image patches. Lipson hand-crafted these flexible templates for a variety of scene classes and showed that they could be used to classify natural scenes of fields, waterfalls and snowy mountains efficiently and reliably. For example, the following concept might be learned for the snowy-mountain class: "if the image contains a blue blob which is above a white blob which is above a brown blob, then it is a mountain". In this paper, we would like to learn such concepts for natural images given a small set of positive and negative examples.

All of the systems described above require users to specify precisely what they want. Minka and Picard [Minka and Picard, 1996] introduced a learning component in their system by using positive and negative examples which let the system choose image groupings within and across images based on color and texture cues; however, their system requires the user to label various parts of the scene, whereas our system only gets a label for the entire image and automatically extracts the relevant parts of the scene. In this paper, we focus on learning natural scene concepts by extracting color and spatial relations between image patches using a small set of positive and negative examples.

Our system uses a small set of user-selected positive and negative examples to learn a scene concept which is used to retrieve similar images from the database. The system also lets the user add more positive and negative examples after each iteration in order to refine the concept.

3 MULTIPLE-INSTANCE LEARNING

In traditional supervised learning, a learning algorithm receives a training set which consists of individually labeled examples. There are situations where this model fails, specifically, when the teacher cannot label individual instances, but only a collection of instances. For example, given a picture containing a waterfall, what is it about the image that causes it to be labeled as a waterfall? Is it the butterfly hovering in the corner, the blooming flowers, or the white stream of water? It is impossible to tell by looking at only one image. The best we can say is that at least one of the objects in the image is a waterfall. Given a number of images (each labeled as waterfall or non-waterfall), we can attempt to find commonalities within the waterfall images that do not appear in the non-waterfall images. Multiple-Instance learning is a way of formalizing this problem, and Diverse Density is a method for finding the commonality.

In Multiple-Instance learning, we receive a set of *bags*, each of which is labeled positive or negative. Each bag contains many *instances*, where each instance is a point in feature space. A bag is labeled negative if all the instances in it are negative. On the other hand, a bag is labeled positive if there is at least one instance in it which is positive. From a collection of labeled bags, the learner tries to induce a concept that will label unseen bags correctly. This problem is harder than even noisy supervised learning because the ratio of negative to positive instances in a positively-labeled bag (the noise ratio) can be arbitrarily high.

The multiple-instance learning model was only recently formalized by [Dietterich *et al.*, 1997], where they develop algorithms for the drug activity prediction problem. This work was followed by [Long and Tan, 1996, Auer *et al.*, 1996, Blum and Kalai, 1998], who showed that it is difficult to PAC-learn in the Multiple-Instance model unless very restrictive independence assumptions are made about the way in which examples are generated. [Auer, 1997] shows that despite these assumptions, the MULTINST algorithm performs competitively on the drug activity

prediction problem. [Maron and Lozano-Pérez, 1998] develop an algorithm called *Diverse Density*, and show that it performs well on a variety of problems such as drug activity prediction, stock selection, and learning a description of a person from a series of images that contain that person.

3.1 MULTIPLE-INSTANCE LEARNING FOR SCENE CLASSIFICATION

In this paper, each training image is a bag. The instances in a particular bag are various subimages. If the bag is labeled as a waterfall (for example), we know that at least one of the subimages (instances) is a waterfall. If the bag is labeled as a non-waterfall, we know that none of the subimages contains a waterfall. Each of the instances, or subimages, is described as a point in some feature space. As discussed in section 4, we experimented with several ways of describing an instance. We will discuss one of them (*single blob with neighbors*) in detail: a subimage is a 2x2 set of pixels (referred to as a *blob*) and its four neighboring blobs (up, down, left, and right). The subimage is described as a vector $[x_1, x_2, \dots, x_{15}]$, where x_1, x_2, x_3 are the mean RGB values of the central blob, x_4, x_5, x_6 are the differences in mean RGB values between the central blob and the blob above it, etc. One bag is therefore a collection of instances, each of which is a point in a 15-dimensional feature space. We assume that at least one of these instances is the template that contains the waterfall.

We would now like to find a description which will correctly classify new images as waterfalls or non-waterfalls. This can be done by finding what is in common between the waterfall images given during training and the differences between those and the non-waterfall images. The main idea behind the *Diverse Density* (DD) algorithm is to find areas in feature space that are close to at least one instance from every positive bag and far from every negative instance. The algorithm searches the feature space for points with high Diverse Density. Once the point (or points) with maximum DD is found, a new image is classified positive if one of its subimages is close to the maximum DD point. As seen in Section 4, the entire database can be sorted by the distance to the learned concept. Figure 1 is a schematic of how the system works.

In the following subsection, we will describe a derivation of Diverse Density and how we find the maximum in a large feature space. We will also show that the appropriate scaling of the feature space can be found by maximizing DD not just with respect to location in

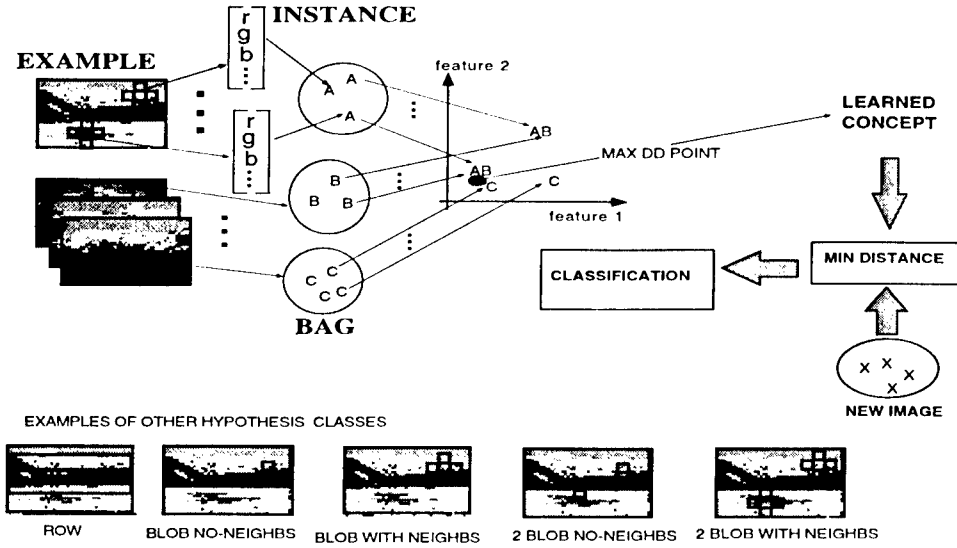


Figure 1: System Diagram

feature space, but also with respect to a weighting of each of the features.

3.2 DIVERSE DENSITY

In this section, we derive a probabilistic measure of Diverse Density. More details are given in [Maron, 1998]. We denote positive bags as B_i^+ , and the j^{th} instance in that bag as B_{ij}^+ . Likewise, B_{ij}^- represents an instance from a negative bag. For simplicity, let us assume that the true concept is a single point t in feature space. We can find t by maximizing $\Pr(t | B_1^+, \dots, B_n^+, B_1^-, \dots, B_m^-)$ over all points in feature space. Using Bayes' rule and a uniform prior over the concept location, we see that this is equivalent to maximizing the likelihood:

$$\arg \max_t \Pr(B_1^+, \dots, B_n^+, B_1^-, \dots, B_m^- | t). \quad (1)$$

By making the additional assumption that the bags are conditionally independent given the target concept t , this decomposes into

$$\arg \max_t \prod_i \Pr(B_i^+ | t) \prod_i \Pr(B_i^- | t) \quad (2)$$

which is equivalent (by similar arguments as above) to maximizing

$$\arg \max_t \prod_i \Pr(t | B_i^+) \prod_i \Pr(t | B_i^-) \quad (3)$$

This is a general definition of Diverse Density, but we need to define the terms in the products to instantiate

it. In this paper, we use the noisy-or model as follows:

$$\Pr(t | B_i^+) = 1 - \prod_j (1 - \Pr(t | B_{ij}^+)). \quad (4)$$

The noisy-or model makes two assumptions: one is that for t to be the target concept it is caused by (hence close to) one of the instances in the bag. It also assumes that the probability of instance j not being the target is independent of any other instance not being the target.

Finally, we estimate the distribution $\Pr(t | B_{ij}^+)$ with a Gaussian-like distribution of $\exp(-\|B_{ij}^+ - t\|^2)$. A negative bag's contribution is likewise computed as $\Pr(t | B_i^-) = \prod_j (1 - \Pr(t | B_{ij}^-))$. A supervised learning algorithm such as nearest-neighbor or kernel regression would average the contribution of each bag, computing a density of instances. This algorithm computes a product of the contribution of each bag, hence the name Diverse Density. Note that Diverse Density at an intersection of n bags is exponentially higher than it is at an intersection of $n - 1$ bags, yet all it takes is one well placed negative instance to drive the Diverse Density down.

The initial feature space is probably not the most suitable one for finding commonalities among images. Some features might be irrelevant or redundant, while small differences along other features might be crucial for discriminating between positive and negative examples. The Diverse Density framework allows us to find the best weighting on the initial feature set in the same way that it allows us to find an appropriate lo-

cation in feature space. If a feature is irrelevant, then removing it can only increase the DD since it will bring positive instances closer together. On the other hand, if a relevant feature is removed then negative instances will come closer to the best DD location and lower it. Therefore, a feature's weight should be changed in order to increase DD. Formally, the distance between two points in feature space (B_{ij} and t) is

$$\|B_{ij}^+ - t\|^2 = \sum_k w_k (B_{ijk} - t_k)^2 \quad (5)$$

where B_{ijk} is the value of the k^{th} feature in the j^{th} point in the i^{th} bag, and w_k is a non-negative scaling factor. If w_k is zero, then the k^{th} feature is irrelevant. If w_k is large, then the k^{th} feature is very important. We would like to find both t and w such that Diverse Density is maximized. We have doubled the number of dimensions in our search space, but we now have a powerful method of changing our representation to accomodate the task.

We can also use this technique to learn more complicated concepts than a single point. To learn a 2-disjunct concept $t \vee s$, we maximize Diverse Density as follows:

$$\arg \max_{t,s} \prod_i (1 - \prod_j (1 - \Pr(t \vee s \mid B_{ij}^+))) \prod_i \prod_j \Pr(t \vee s \mid B_{ij}^-) \quad (6)$$

where $\Pr(t \vee s \mid B_{ij}^+)$ is estimated as $\max\{\Pr(t \mid B_{ij}^+), \Pr(s \mid B_{ij}^+)\}$. Other approximations (such as noisy-or) are also possible.

Finding the maximum Diverse Density in a high-dimensional space is a difficult problem. In general, we are searching an arbitrary landscape and the number of local maxima and size of the search space could prohibit any efficient exploration. In this paper, we use gradient ascent (since DD is a differentiable function) with multiple starting points. This has worked successfully because we know what starting points to use. The maximum DD point is made of contributions from some set of positive points. If we start an ascent from every positive point, one of them is likely to be closest to the maximum, contribute the most to it and have a climb directly to it. Therefore, if we start an ascent from every positive instance, we are very likely to find the maximum DD point. When we need to find both the location and the scaling of the concept, we perform gradient ascent for both sets of parameters at the same time (starting with all scale weightings at

1). The number of dimensions in our search space has doubled, though. When we need to find a 2-disjunct concept, we can again perform gradient ascent for all parameters at once. This carries a high computational burden because the number of dimensions has doubled, and we perform a gradient ascent starting at every pair of positive instances.

Our goal in the next section is to show that: (1) Multiple-Instance learning by maximizing diverse density can be used in the domain of natural scene classification, (2) simple concepts in low resolution images are sufficient to learn some of these concepts (3) adding false positives and false negatives over multiple iterations (user interaction) can be used to improve the classifier performance.

4 EXPERIMENTS

In this section, we show four different types of results from running the system: one is that Multiple-Instance learning is applicable to this domain. A second result is that one does not need very complicated hypothesis classes to learn concepts from the natural image domain. We also compare the performance of various hypotheses, including the global histogram method. Finally, we show how user interaction would work to improve the classifier.

4.1 EXPERIMENTAL SETUP

We tried to learn three different concepts: waterfall, mountain, and field. For training and testing we used natural images from the COREL library, and the labels given by COREL. These included 100 images from each of the following classes: waterfalls, fields, mountains, sunsets and lakes. We also used a larger test set of 2600 natural images from various classes.

We created a *potential training set* that consisted of 20 randomly chosen images from each of the five classes mentioned above. This left us with a *small test set* consisting of the remaining 80 images from each of the five classes. We separated the potential training set from the testing set to insure that results of using various training schemes and hypothesis classes can be compared fairly. Finally the *large test set* contained 2600 natural images from a large variety of classes.

For a given concept, we create an *initial training set* by picking five positive examples of the concept and five negative examples, all from the potential training set. After the concept is learned from these examples (by finding the point in and scaling of feature

space with maximum DD), the unused 90 images in the potential training set are sorted by distance from the learned concept¹. This sorted list can be used to simulate what a user would select as further refining examples. Specifically, the most egregious false positives (the non-concept images at the beginning of the sorted list) and the most egregious false negatives (the concept images at the end of the sorted list) would likely be picked by the user as additional negative and positive examples.

We attempted four different training schemes: *initial* is simply using the initial five positives and five negative examples. *+5fp* adds the five most egregious false positives. *+10fp* repeats the *+5fp* scheme twice. *+3fp+2fn* adds 3 false positives and 2 false negatives.

All images were smoothed using a gaussian filter and subsampled to 8×8 . We used the RGB color space in these experiments. For every class and for every training scheme, we tried to learn the concept using one of seven hypothesis classes (Figure 1 shows some examples):

1. *row*: an instance is the row's mean color and the color difference in the rows above and below it.
2. *single blob with neighbors*: an instance is the mean color of a 2×2 blob and the color difference with its 4 neighboring blobs.
3. *single blob with no neighbors*: an instance is the color of each of the pixels in a 2×2 blob.
4. *disjunctive blob with neighbors*: an instance is the same as the single blob with neighbors but the concept learned is a disjunction of two single blob concepts.
5. *disjunctive blob with no neighbors*: an instance is the same as the single blob with no neighbors but the concept learned is a disjunction of two single blob concepts.
6. *two blob with neighbors*: an instance is the mean color of two descriptions of two *single blob with neighbors* and their relative spatial relationship (whether the second blob is above or below, and whether it is to the left or right, of the first blob).
7. *two blob with no neighbors*: an instance is the mean color of two descriptions of two *single blob with no neighbors* and their relative spatial relationship.

Learning a concept took anywhere from a few sec-

¹An image/bag's distance from the concept is the minimum distance of any of the image's subregions/instances from the concept.

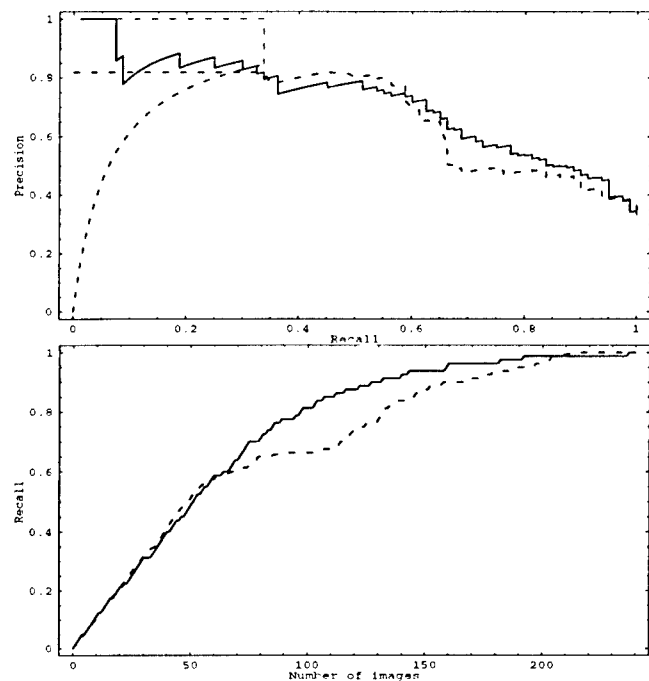


Figure 2: Comparison of learned concept (solid curves) with hand-crafted templates (dashed curves) for the mountain concept on 240 images from the small test set. The top and bottom dashed precision-recall curves indicate the best-case and worst-case curves for the first 32 images retrieved by the hand-crafted template which all have the same score.

onds for the simple hypotheses to a few days for the 2-blob and disjunctive hypotheses. The more complicated hypotheses take longer to learn because of the higher number of features and because the number of instances per bag is large (and to find the maximum DD point, we perform a gradient ascent from every positive instance). Because this is a prototype, we have not tried to optimize the running time; however, a more intelligent method of generating instances (for example, a rough segmentation using connected components) will reduce both the number of instances and the running time by orders of magnitude.

4.2 RESULTS

In this section we show results of testing the various hypothesis classes, training schemes, and concept classes against the small test set and the larger one. The small test set does not intersect the potential training set, and therefore more accurately represents the generalization of the learned concepts. The large test set is meant to show how the system scales to larger image databases.

The graphs shown are precision-recall and recall curves. Precision is the ratio of the number of correct images to the number of images seen so far. Recall is the ratio of the number of correct images to the total number of correct images in the test set. For example, in Figure 3, the waterfall precision-recall curve has recall 0.5 with precision of about 0.7, which means in order to retrieve 40 of the 80 waterfalls, 30% of the images retrieved are not waterfalls. We show both curves for because (1) the beginning of the precision-recall is of interest to applications where only the top few objects are of importance, and (2) the middle of the recall curve is of interest to applications where correct classification of a large percentage of the database is important.

Figure 2 shows that the performance of the learned mountain concept is competitive with a hand-crafted mountain template (from [Lipson *et al.*, 1997]²). The test set consists of 80 mountains, 80 fields, and 80 waterfalls. It is disjoint from the training set. The hand-crafted model's precision-recall curve is flat at 84% because the first 32 images all receive the same score, and 27 of them are mountains. We also show the curves if we were to retrieve the 27 mountains first (best-case) or after the first five false positives (worst-case).

In Figure 3, we show the performance of the best hypothesis and training method on each concept class. The dashed lines show the poor performance of the global histogram method. The solid lines in the precision-recall graph show the performance of single blob with neighbors with +10fp for waterfalls, row with +10fp for fields, and disjunctive blob with no neighbors with +10fp for mountains. The solid lines in the recall curve show the performance of the single blob with neighbors with +10fp for waterfalls, single blob with neighbors with +3fp+2fn for fields, and row with +3fp+2fn for mountains. This behavior continues for the larger test set.

In Figure 4, we show the precision-recall curves for each of the four training schemes. We average over all concepts and all hypothesis classes. We see that performance improves with user interaction. This behavior continues for the larger test set as well.

In Figure 5, we show the precision-recall and recall curves for each of the seven hypotheses averaged over all concepts and all training schemes. Note that these curves are for the larger 2600 image database. We

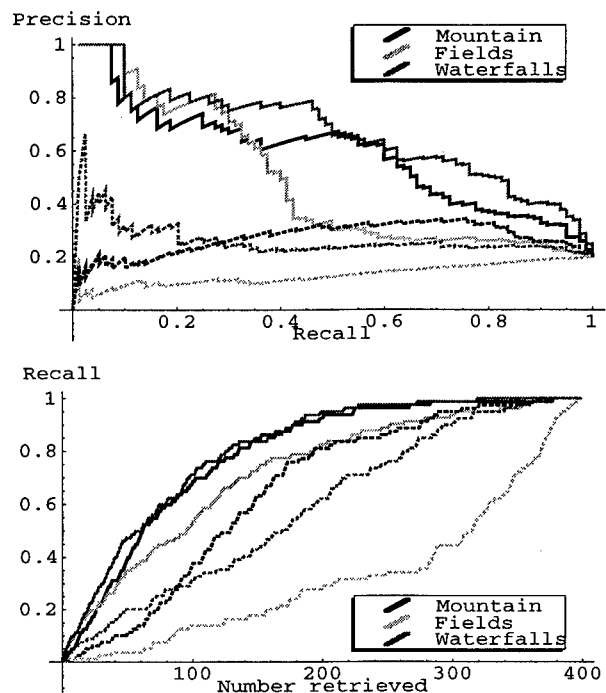


Figure 3: The best curves for each concept using a small test set. Dashed curves are the global histogram's performance.

see that the single blob with neighbors hypothesis has good precision. We also see that the more complicated hypothesis classes (i.e. the disjunctive concepts and the two-blob concepts) tend to have better recall curves.

In Figure 6, we show a snapshot of the system in action. The system is trained using training scheme +10fp for the waterfall concept. It has learned a waterfall concept using the single blob with neighbors hypothesis. The learned waterfall concept is that somewhere in the image there is a blob whose left neighbor is less blue, whose own blue value is 0.5 (where RGB values are in the $[0, 1]$ cube), whose neighbor below has the same blue value, whose neighbor above has the same red value, whose green value is 0.55, whose neighbor above has the same blue value and whose red value is 0.47. These properties are weighted in the order given, and any other features were found to be irrelevant. A new image has the rating of the minimum distance of one of its instances to the learned concept, where the distance metric uses the learned scaling to account for the importance of the relevant features. As we can see in the figure, this simple learned concept is able to retrieve a wide variety

²Lipson's classifier was modified to give a ranking of each image, rather than its class.

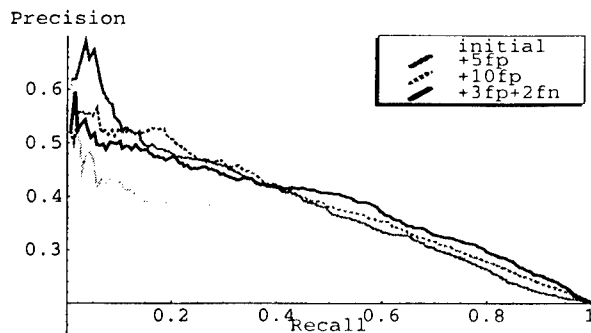


Figure 4: Different training schemes, averaged over concept and hypothesis class, using a small test set.

of waterfall scenes.

The top 20 images in the figure are the training set. The first 10 images are the initial positive and negative examples used in training. The next 10 images are the false positives added. The last 30 images are the top 30 returned from the large dataset.

5 CONCLUSIONS

In this paper, we have shown that Multiple-Instance learning by maximizing diverse density can be used to classify images of natural scenes. Our results are competitive with hand-crafted models, and much better than a global histogram approach. We have also demonstrated that simple learned concepts that capture color relations in low resolution images can be used effectively in the domain of natural scene classification. Our experiments indicate that complicated concepts (e.g. disjunctive concepts) tend to have better recall curves and that user interaction (adding false positives and false negatives) over multiple iterations can improve the performance of the classifier. Our architecture, by separating the bag generator from the learning mechanism, allows progress in the field of computer vision to benefit the field of machine learning and vice versa.

Acknowledgements

We thank Tomás Lozano-Pérez, Eric Grimson, and Pam Lipson for their advice and AFOSR ASSERT program Parent Grant#:F49620-93-1-0263, and ARPA under ONR contract N00014-95-1-0600 for their support of this research.

References

- [Auer *et al.*, 1996] Peter Auer, Phil M. Long, and A. Sriniwasan. Approximating hyper-rectangles: learning and pseudorandom sets. In *Proceedings of the 1996 Conference on Computational Learning Theory*, 1996.
- [Auer, 1997] Peter Auer. On Learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proceedings of the 14th International Conference on Machine Learning*, 1997.
- [Belongie *et al.*, 1998] S. Belongie, C. Carson, H. Greenspan, and J. Malik. Color- and Texture based image segmentation using EM and its application to content-based image retrieval. In *International Conference on Computer Vision*, 1998.
- [Blum and Kalai, 1998] A. Blum and A. Kalai. A Note on Learning from Multiple-Instance Examples. *To appear in Machine Learning*, 1998.
- [Dietterich *et al.*, 1997] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the Multiple-Instance Problem with Axis-Parallel Rectangles. *Artificial Intelligence Journal*, 89, 1997.
- [Flickner *et al.*, 1995] M. Flickner, , and et al. Query by image and video content: The QBIC System. *IEEE Computer*, 28:23-32, 1995.
- [Huang *et al.*, 1997] J. Huang, S. Ravikumar, M. Mitra, W. Zhu, and R. Zabih. Image indexing using color correlograms. In *Computer Vision and Pattern Recognition*, 1997.
- [Keeler *et al.*, 1991] James D. Keeler, David E. Rumelhart, and Wee-Kheng Leow. Integrated Segmentation and Recognition of Hand-Printed Numerals. In *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, 1991.
- [Lipson *et al.*, 1997] P. Lipson, E. Grimson, and P. Sinha. Context and Configuration Based Scene Classification. In *Computer Vision and Pattern Recognition*, 1997.
- [Long and Tan, 1996] P. M. Long and L. Tan. PAC-learning axis aligned rectangles with respect to product distributions from multiple-instance examples. In *Proceedings of the 1996 Conference on Computational Learning Theory*, 1996.
- [Maron and Lozano-Pérez, 1998] O. Maron and T. Lozano-Pérez. A framework for Multiple-Instance learning. In *Advances in Neural Information Processing Systems 10*. MIT Press, 1998.
- [Maron, 1998] O. Maron. Learning from Ambiguity. Doctoral Thesis, Dept. of Electrical Engineering and Computer Science, M.I.T., June 1998.
- [Minka and Picard, 1996] T. Minka and R. Picard. Interactive Learning using a society of models. In *Computer Vision and Pattern Recognition*, 1996.
- [Rubner *et al.*, 1998] Y. Rubner, C. Tomasi, and L. Guibas. A Metric for Distributions with Applications to Image Databases. In *Proceedings of IEEE Int. Conf. on Computer Vision*, 1998.
- [Smith and Chang, 1996] J. Smith and S. Chang. VisualSEEK: a fully automated content-based image query system. In *Proc. ACM International Conference on Multimedia*. Morgan Kaufmann, 1996.

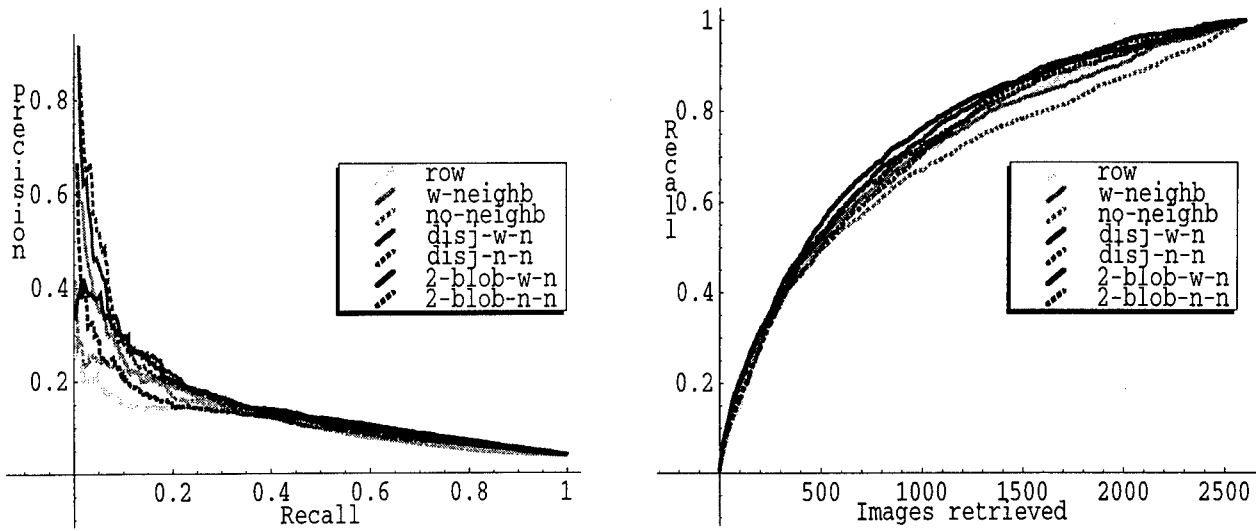


Figure 5: Different hypothesis classes averaged over concept and training scheme, using a large test set with 2600 images.

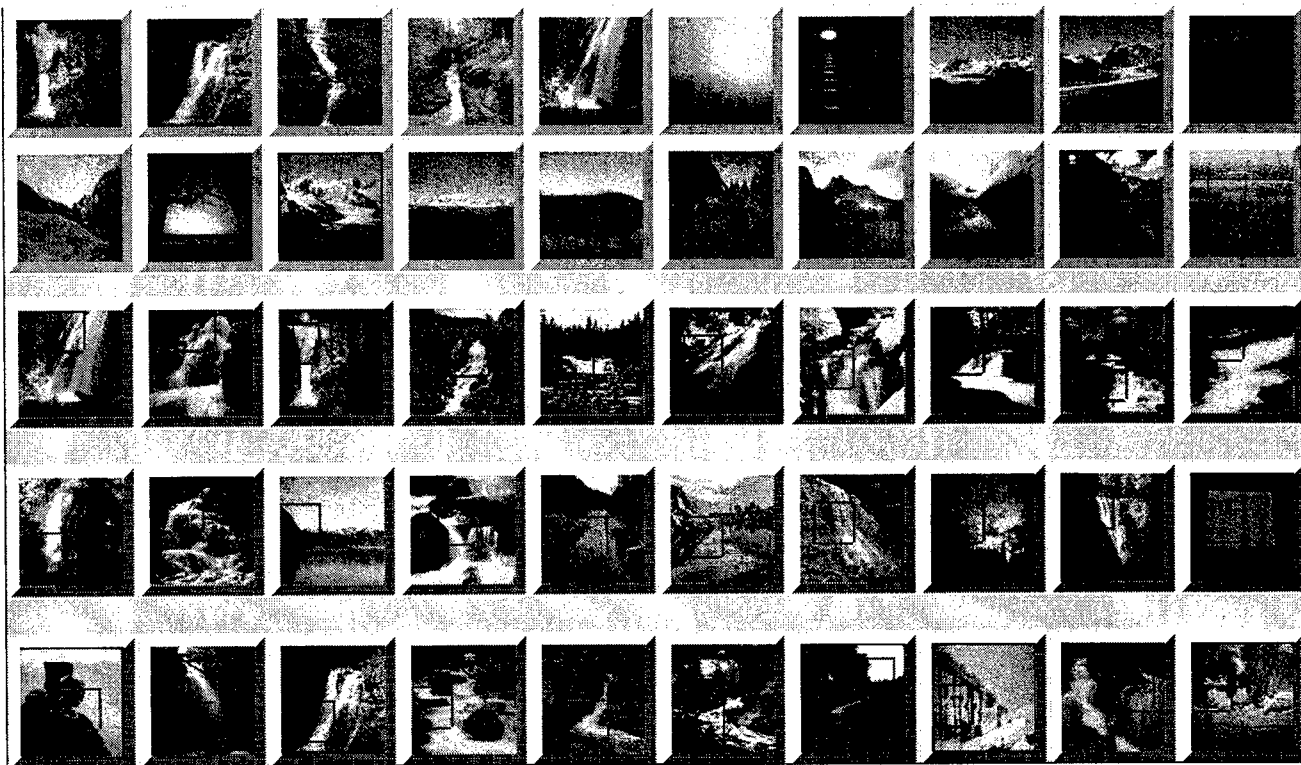


Figure 6: Results for the waterfall concept using the single blob with neighbors concept with +10fp. Top row: Initial training set-5 positive and 5 negative examples. Second Row: Additional false positives. Last three rows: Top 30 matches retrieved from the large test set. The red squares indicate where the closest instance to the learned concept is located.

Employing EM and Pool-Based Active Learning for Text Classification

Andrew Kachites McCallum^{††}
mccallum@justresearch.com

[†]Just Research
4616 Henry Street
Pittsburgh, PA 15213

Kamal Nigam[†]
knigam@cs.cmu.edu

[†]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper shows how a text classifier's need for labeled training documents can be reduced by taking advantage of a large pool of unlabeled documents. We modify the Query-by-Committee (QBC) method of active learning to use the unlabeled pool for explicitly estimating document density when selecting examples for labeling. Then active learning is combined with Expectation-Maximization in order to "fill in" the class labels of those documents that remain unlabeled. Experimental results show that the improvements to active learning require less than two-thirds as many labeled training examples as previous QBC approaches, and that the combination of EM and active learning requires only slightly more than half as many labeled training examples to achieve the same accuracy as either the improved active learning or EM alone.

1 Introduction

Obtaining labeled training examples for text classification is often expensive, while gathering large quantities of unlabeled examples is usually very cheap. For example, consider the task of learning which web pages a user finds interesting. The user may not have the patience to hand-label a thousand training pages as interesting or not, yet multitudes of unlabeled pages are readily available on the Internet.

This paper presents techniques for using a large pool of unlabeled documents to improve text classification when labeled training data is sparse. We enhance the

QBC active learning algorithm to select labeling requests from the entire pool of unlabeled documents, and explicitly use the pool to estimate regional document density. We also combine active learning with Expectation-Maximization (EM) in order to take advantage of the word co-occurrence information contained in the many documents that remain in the unlabeled pool.

In previous work [Nigam *et al.* 1998] we show that combining the evidence of labeled and unlabeled documents via EM can reduce text classification error by one-third. We treat the absent labels as "hidden variables" and use EM to fill them in. EM improves the classifier by alternately using the current classifier to guess the hidden variables, and then using the current guesses to advance classifier training. EM consequently finds the classifier parameters that locally maximize the probability of both the labeled and unlabeled data.

Active learning approaches this same problem in a different way. Unlike our EM setting, the active learner can request the true class label for certain unlabeled documents it selects. However, each request is considered an expensive operation and the point is to perform well with as few queries as possible. Active learning aims to select the most informative examples—in many settings defined as those that, if their class label were known, would maximally reduce classification error and variance over the distribution of examples [Cohn, Ghahramani, & Jordan 1996]. When calculating this in closed-form is prohibitively complex, the *Query-by-Committee* (QBC) algorithm [Freund *et al.* 1997] can be used to select documents that have high classification variance themselves. QBC measures the variance indirectly, by examining the disagreement among class labels assigned by a set of classifier variants, sampled from the probability distribution of clas-

sifiers that results from the labeled training examples.

This paper shows that a pool of unlabeled examples can be used to benefit both active learning and EM. Rather than having active learning choose queries by synthetically generating them (which is awkward with text), or by selecting examples from a stream (which inefficiently models the data distribution), we advocate selecting the best examples from the entire pool of unlabeled documents (and using the pool to explicitly model density); we call this last scheme *pool-based sampling*. In experimental results on a real-world text data set, this technique is shown to reduce the need for labeled documents by 42% over previous QBC approaches. Furthermore, we show that the combination of QBC and EM learns with fewer labeled examples than either individually—requiring only 58% as many labeled examples as EM alone, and only 26% as many as QBC alone. We also discuss our initial approach to a richer combination we call *pool-leveraged sampling* that interleaves active learning and EM such that EM's modeling of the unlabeled data informs the selection of active learning queries.

2 Probabilistic Framework for Text Classification

This section presents a Bayesian probabilistic framework for text classification. The next two sections add EM and active learning by building on this framework. We approach the task of text classification from a Bayesian learning perspective: we assume that the documents are generated by a particular parametric model, and use training data to calculate Bayes-optimal estimates of the model parameters. Then, we use these estimates to classify new test documents by turning the generative model around with Bayes' rule, calculating the probability that each class would have generated the test document in question, and selecting the most probable class.

Our parametric model is naive Bayes, which is based on commonly used assumptions [Friedman 1997; Joachims 1997]. First we assume that text documents are generated by a mixture model (parameterized by θ), and that there is a one-to-one correspondence between the (observed) class labels and the mixture components. We use the notation $c_j \in \mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$ to indicate both the j th component and j th class. Each component c_j is parameterized by a disjoint subset of θ . These assumptions specify that a document is created by (1) selecting a class according to the prior probabilities, $P(c_j|\theta)$, then (2) having that class com-

ponent generate a document according to its own parameters, with distribution $P(d_i|c_j; \theta)$. We can characterize the likelihood of a document as a sum of total probability over all generative components:

$$P(d_i|\theta) = \sum_{j=1}^{|\mathcal{C}|} P(c_j|\theta)P(d_i|c_j; \theta). \quad (1)$$

Document d_i is considered to be an ordered list of word events. We write $w_{d_{ik}}$ for the word in position k of document d_i , where the subscript of w indicates an index into the vocabulary $V = \langle w_1, w_2, \dots, w_{|V|} \rangle$. We make the standard naive Bayes assumption: that the words of a document are generated independently of context, that is, independently of the other words in the same document given the class. We further assume that the probability of a word is independent of its position within the document. Thus, we can express the class-conditional probability of a document by taking the product of the probabilities of the independent word events:

$$P(d_i|c_j; \theta) = P(|d_i|) \prod_{k=1}^{|d_i|} P(w_{d_{ik}}|c_j; \theta), \quad (2)$$

where we assume the length of the document, $|d_i|$, is distributed independently of class. Each individual class component is parameterized by the collection of word probabilities, such that $\theta_{w_t|c_j} = P(w_t|c_j; \theta)$, where $t \in \{1, \dots, |V|\}$ and $\sum_t P(w_t|c_j; \theta) = 1$. The other parameters of the model are the class prior probabilities $\theta_{c_j} = P(c_j|\theta)$, which indicate the probabilities of selecting each mixture component.

Given these underlying assumptions of how the data are produced, the task of learning a text classifier consists of forming an estimate of θ , written $\hat{\theta}$, based on a set of training data. With labeled training documents, $\mathcal{D} = \{d_1, \dots, d_{|\mathcal{D}|}\}$, we can calculate estimates for the parameters of the model that generated these documents. To calculate the probability of a word given a class, $\theta_{w_t|c_j}$, simply count the fraction of times the word occurs in the data for that class, augmented with a Laplacean prior. This smoothing prevents probabilities of zero for infrequently occurring words. These word probability estimates $\hat{\theta}_{w_t|c_j}$ are:

$$\hat{\theta}_{w_t|c_j} = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} N(w_t, d_i)P(c_j|d_i)}{|V| + \sum_{s=1}^{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{D}|} N(w_s, d_i)P(c_j|d_i)}, \quad (3)$$

where $N(w_t, d_i)$ is the count of the number of times word w_t occurs in document d_i , and where $P(c_j|d_i) \in \{0, 1\}$, given by the class label. The class prior probabilities, $\hat{\theta}_{c_j}$, are estimated in the same fashion of counting, but without smoothing:

$$\hat{\theta}_{c_j} = \frac{\sum_{i=1}^{|\mathcal{D}|} P(c_j|d_i)}{|\mathcal{D}|}. \quad (4)$$

Given estimates of these parameters calculated from the training documents, it is possible to turn the generative model around and calculate the probability that a particular class component generated a given document. We formulate this by an application of Bayes' rule, and then substitutions using Equations 1 and 2:

$$P(c_j|d_i; \hat{\theta}) = \frac{P(c_j|\hat{\theta}) \prod_{k=1}^{|d_i|} P(w_{d_{i,k}}|c_j; \hat{\theta})}{\sum_{r=1}^{|\mathcal{C}|} P(c_r|\hat{\theta}) \prod_{k=1}^{|d_i|} P(w_{d_{i,k}}|c_r; \hat{\theta})}. \quad (5)$$

If the task is to classify a test document d_i into a single class, simply select the class with the highest posterior probability: $\arg \max_j P(c_j|d_i; \hat{\theta})$.

Note that our assumptions about the generation of text documents are all violated in practice, and yet empirically, naive Bayes does a good job of classifying text documents [Lewis & Ringuette 1994; Craven *et al.* 1998; Joachims 1997]. This paradox is explained by the fact that classification estimation is only a function of the sign (in binary cases) of the function estimation [Friedman 1997; Domingos & Pazzani 1997]. Also note that our formulation of naive Bayes assumes a multinomial event model for documents; this generally produces better text classification accuracy than another formulation that assumes a multi-variate Bernoulli [McCallum & Nigam 1998].

3 EM and Unlabeled Data

When naive Bayes is given just a small set of labeled training data, classification accuracy will suffer because variance in the parameter estimates of the generative model will be high. However, by augmenting this small set with a large set of unlabeled data and combining the two pools with EM, we can improve the parameter estimates. This section describes how to use EM to combine these pools within the probabilistic framework of the previous section.

EM is a class of iterative algorithms for maximum likelihood estimation in problems with incomplete data

[Dempster, Laird, & Rubin 1977]. Given a model of data generation, and data with some missing values, EM alternately uses the current model to estimate the missing values, and then uses the missing value estimates to improve the model. Using all the available data, EM will locally maximize the likelihood of the generative parameters, giving estimates for the missing values.

In our text classification setting, we treat the class labels of the unlabeled documents as missing values, and then apply EM. The resulting naive Bayes parameter estimates often give significantly improved classification accuracy on the test set when the pool of labeled examples is small [Nigam *et al.* 1998].¹ This use of EM is a special case of a more general missing values formulation [Ghahramani & Jordan 1994].

In implementation, EM is an iterative two-step process. The E-step calculates probabilistically-weighted class labels, $P(c_j|d_i; \hat{\theta})$, for every unlabeled document using a current estimate of θ and Equation 5. The M-step calculates a new maximum likelihood estimate for θ using all the labeled data, both original and probabilistically labeled, by Equations 3 and 4. We initialize the process with parameter estimates using just the labeled training data, and iterate until $\hat{\theta}$ reaches a fixed point. See [Nigam *et al.* 1998] for more details.

4 Active Learning with EM

Rather than estimating class labels for unlabeled documents, as EM does, active learning instead requests the *true* class labels for unlabeled documents it selects. In many settings, an optimal active learner should select those documents that, when labeled and incorporated into training, will minimize classification error over the distribution of future documents. Equivalently in probabilistic frameworks without bias, active learning aims to minimize the expected classification variance over the document distribution. Note that Naive Bayes' independence assumption and Laplacean priors do introduce bias. However, variance tends to dominate bias in classification error [Friedman 1997], and thus we focus on reducing variance.

The Query-by-Committee (QBC) method of active learning measures this variance indirectly [Freund *et al.* 1997]. It samples several times from the classifier parameter distribution that results from the training

¹When the classes do not correspond to the natural clusters of the data, EM can hurt accuracy instead of helping. Our previous work also describes a method for avoiding these detrimental effects.

data, in order to create a “committee” of classifier variants. This committee approximates the entire classifier distribution. QBC then classifies unlabeled documents with each committee member, and measures the disagreement between their classifications—thus approximating the classification variance. Finally, documents on which the committee disagrees strongly are selected for labeling requests. The newly labeled documents are included in the training data, and a new committee is sampled for making the next set of requests. This section presents each of these steps in detail, and then explains its integration with EM. Our implementation of this algorithm is summarized in Table 1.

Our committee members are created by sampling classifiers according to the distribution of classifier parameters specified by the training data. Since the probability of the naive Bayes parameters for each class are described by a Dirichlet distribution, we sample the parameters $\theta_{w_t|c_j}$ from the posterior Dirichlet distribution based on training data word counts, $N(\cdot, \cdot)$. This is performed by drawing weights, v_{tj} , for each word w_t and class c_j from the Gamma distribution: $v_{tj} = \text{Gamma}(\alpha_t + N(w_t, c_j))$, where α_t is always 1, as specified by our Laplacean prior. Then we set the parameters $\theta_{w_t|c_j}$ to the normalized weights by $\theta_{w_t|c_j} = v_{tj} / \sum_s v_{sj}$. We sample to create a classifier k times, resulting in k committee members. Individual committee members are denoted by m .

We consider two metrics for measuring committee disagreement. The previously employed *vote entropy* [Dagan & Engelson 1995] is the entropy of the class label distribution resulting from having each committee member “vote” with probability mass $1/k$ for its winning class. One disadvantage of vote entropy is that it does not consider the confidence of the committee members’ classifications, as indicated by the class probabilities $P_m(c_j|d_i; \hat{\theta})$ from each member.

To capture this information, we propose to measure committee disagreement for each document using *Kullback-Leibler divergence to the mean* [Pereira, Tishby, & Lee 1993]. Unlike vote entropy, which compares only the committee members’ top ranked class, KL divergence measures the strength of the certainty of disagreement by calculating differences in the committee members’ class distributions, $P_m(C|d_i)$.² Each

²While naive Bayes is not an accurate probability estimator [Domingos & Pazzani 1997], naive Bayes classification scores are somewhat correlated to confidence; the fact that naive Bayes scores can be successfully used to make accuracy/coverage trade-offs is testament to this.

-
- Calculate the density for each document. (Eq. 9)
 - Loop while adding documents:
 - Build an initial estimate of $\hat{\theta}$ from the labeled documents only. (Eqs. 3 and 4)
 - Loop k times, once for each committee member:
 - + Create a committee member by sampling for each class from the appropriate Dirichlet distribution.
 - + *Starting with the sampled classifier apply EM with the unlabeled data. Loop while parameters change:*
 - *Use the current classifier to probabilistically label the unlabeled documents. (Eq. 5)*
 - *Recalculate the classifier parameters given the probabilistically-weighted labels. (Eqs. 3 and 4)*
 - + Use the current classifier to probabilistically label all unlabeled documents. (Eq. 5)
 - Calculate the disagreement for each unlabeled document (Eq. 7), multiply by its density, and request the class label for the one with the highest score.
 - Build a classifier with the labeled data. (Eqs. 3 and 4).
 - *Starting with this classifier, apply EM as above.*
-

Table 1: Our active learning algorithm. Traditional Query-by-Committee omits the EM steps, indicated by italics, does not use the density, and works in a stream-based setting.

committee member m produces a posterior class distribution, $P_m(C|d_i)$, where C is a random variable over classes. KL divergence to the mean is an average of the KL divergence between each distribution and the mean of all the distributions:

$$\frac{1}{k} \sum_{m=1}^k D(P_m(C|d_i) || P_{avg}(C|d_i)), \quad (6)$$

where $P_{avg}(C|d_i)$ is the class distribution mean over all committee members, m : $P_{avg}(C|d_i) = (\sum_m P_m(C|d_i))/k$.

KL divergence, $D(\cdot || \cdot)$, is an information-theoretic measure of the difference between two distributions, capturing the number of extra “bits of information” required to send messages sampled from the first distribution using a code that is optimal for the second. The KL divergence between distributions $P_1(C)$ and $P_2(C)$ is:

$$D(P_1(C) || P_2(C)) = \sum_{j=1}^{|C|} P_1(c_j) \log \left(\frac{P_1(c_j)}{P_2(c_j)} \right). \quad (7)$$

After disagreement has been calculated, a document is selected for a class label request. (Selecting more than one document at a time can be a computational convenience.) We consider three ways of selecting documents: stream-based, pool-based, and density-weighted pool-based. Some previous applications of QBC [Dagan & Engelson 1995; Liere & Tadepalli 1997] use a simulated stream of unlabeled documents. When a document is produced by the stream, this approach measures the classification disagreement among the committee members, and decides, based on the disagreement, whether to select that document for labeling. Dagan and Engelson do this heuristically by dividing the vote entropy by the maximum entropy to create a probability of selecting the document. Disadvantages of using *stream-based sampling* are that it only sparsely samples the full distribution of possible document labeling requests, and that the decision to label is made on each document individually, irrespective of the alternatives.

An alternative that aims to address these problems is *pool-based sampling*. It selects from among all the unlabeled documents in a pool the one with the largest disagreement. However, this loses one benefit of stream-based sampling—the implicit modeling of the data distribution—and it may select documents that have high disagreement, but are in unimportant, sparsely populated regions.

We can retain this distributional information by selecting documents using both the classification disagreement and the “density” of the region around a document. This *density-weighted pool-based sampling* method prefers documents with high classification variance that are also similar to many other documents. The stream approach approximates this implicitly; we accomplish this more accurately, (especially when labeling a small number of documents), by modeling the density explicitly.

We approximate the density in a region around a particular document by measuring the average distance from that document to all other documents. Distance, Y , between individual documents is measured by using exponentiated KL divergence:

$$Y(d_i, d_h) = e^{-\beta D(P(W|d_h) \parallel (\lambda P(W|d_i) + (1-\lambda)P(W)))}, \quad (8)$$

where W is a random variable over words in the vocabulary; $P(W|d_i)$ is the maximum likelihood estimate of words sampled from document d_i , (i.e.,

$P(w_t|d_i) = N(w_t, d_i)/|d_i|$); $P(W)$ is the marginal distribution over words; λ is a parameter that determines how much smoothing to use on the encoding distribution (we must ensure no zeroes here to prevent infinite distances); and β is a parameter that determines the sharpness of the distance metric.

In essence, the average KL divergence between a document, d_i , and all other documents measures the degree of overlap between d_i and all other documents; exponentiation converts this information-theoretic number of “bits of information” into a scalar distance.

When calculating the average distance from d_i to all other documents it is much more computationally efficient to calculate the geometric mean than the arithmetic mean, because the distance to all documents that share no words with d_i can be calculated in advance, and we only need make corrections for the words that appear in d_i . Using a geometric mean, we define density, Z of document d_i to be

$$Z(d_i) = e^{\frac{1}{|\mathcal{D}|} \sum_{d_h \in \mathcal{D}} \ln(Y(d_i, d_h))}. \quad (9)$$

We combine this density metric with disagreement by selecting the document that has the largest product of density (Equation 9) and disagreement (Equation 6). This density-weighted pool-based sampling selects the document that is representative of many other documents, and about which there is confident committee disagreement.

Combining Active Learning and EM

Active learning can be combined with EM by running EM to convergence after actively selecting all the training data that will be labeled. This can be understood as using active learning to select a better starting point for EM hill climbing, instead of randomly selecting documents to label for the starting point. A more interesting approach, that we term *pool-leveraged sampling*, is to interleave EM with active learning, so that EM not only builds on the results of active learning, but EM also informs active learning. To do this we run EM to convergence on each committee member before performing the disagreement calculations. The intended effect is (1) to avoid requesting labels for examples whose label can be reliably filled in by EM, and (2) to encourage the selection of examples that will help EM find a local maximum with higher classification accuracy. With more accurate committee members, QBC should pick more informative documents to label. The complete active learning algo-

rithm, both with and without EM, is summarized in Table 1.

Unlike settings in which queries must be generated [Cohn 1994], and previous work in which the unlabeled data is available as a stream [Dagan & Engelson 1995; Liere & Tadepalli 1997; Freund *et al.* 1997], our assumption about the availability of a pool of unlabeled data makes the improvements to active learning possible. This pool is present for many real-world tasks in which efficient use of labels is important, especially in text learning.

5 Related Work

A similar approach to active learning, but without EM, is that of Dagan and Engelson [1995]. They use QBC stream-based sampling and vote entropy. In contrast, we advocate density-weighted pool-based sampling and the KL metric. Additionally, we select committee members using the Dirichlet distribution over classifier parameters, instead of approximating this with a Normal distribution. Several other studies have investigated active learning for text categorization. Lewis and Gale examine uncertainty sampling and relevance sampling in a pool-based setting [Lewis & Gale 1994; Lewis 1995]. These techniques select queries based on only a single classifier instead of a committee, and thus cannot approximate classification variance. Liere and Tadepalli [1997] use committees of Winnow learners for active text learning. They select documents for which two randomly selected committee members disagree on the class label.

In previous work, we show that EM with unlabeled data reduces text classification error by one-third [Nigam *et al.* 1998]. Two other studies have used EM to combine labeled and unlabeled data without active learning for classification, but on non-text tasks [Miller & Uyar 1997; Shahshahani & Landgrebe 1994]. Ghahramani and Jordan [1994] use EM with mixture models to fill in missing feature values.

6 Experimental Results

This section provides evidence that using a combination of active learning and EM performs better than using either individually. The results are based on data sets from UseNet and Reuters.³

³These data sets are both available on the Internet. See <http://www.cs.cmu.edu/~textlearning> and <http://www.research.att.com/~lewis>.

The Newsgroups data set, collected by Ken Lang, contains about 20,000 articles evenly divided among 20 UseNet discussion groups [Joachims 1997]. We use the five comp.* classes as our data set. When tokenizing this data, we skip the UseNet headers (thereby discarding the subject line); tokens are formed from contiguous alphabetic characters, removing words on a stoplist of common words. Best performance was obtained with no feature selection, no stemming, and by normalizing word counts by document length. The resulting vocabulary, after removing words that occur only once, has 22958 words. On each trial, 20% of the documents are randomly selected for placement in the test set.

The 'ModApte' train/test split of the Reuters 21578 Distribution 1.0 data set consists of 12902 Reuters newswire articles in 135 overlapping topic categories. Following several other studies [Joachims 1998; Liere & Tadepalli 1997] we build binary classifiers for each of the 10 most populous classes. We ignore words on a stoplist, but do not use stemming. The resulting vocabulary has 19371 words. Results are reported on the complete test set as precision-recall breakeven points, a standard information retrieval measure for binary classification [Joachims 1998].

In our experiments, an initial classifier was trained with one randomly-selected labeled document per class. Active learning proceeds as described in Table 1. Newsgroups experiments were run for 200 active learning iterations, each round selecting one document for labeling. Reuters experiments were run for 100 iterations, each round selecting five documents for labeling. Smoothing parameter λ is 0.5; sharpness parameter β is 3. We made little effort to tune β and none to tune λ . For QBC we use a committee size of three ($k=3$); initial experiments show that committee size has little effect. All EM runs perform seven EM iterations; we never found classification accuracy to improve beyond the seventh iteration. All results presented are averages of ten runs per condition.

The top graph in Figure 1 shows a comparison of different disagreement metrics and selection strategies for QBC without EM. The best combination, density-weighted pool-based sampling with a KL divergence to the mean disagreement metric achieves 51% accuracy after acquiring only 30 labeled documents. To reach the same accuracy, unweighted pool-based sampling with KL disagreement needs 40 labeled documents. If we switch to stream-based, sampling, KL disagreement needs 51 labelings for 51% accuracy. Our random selection baseline requires 59 labeled documents.

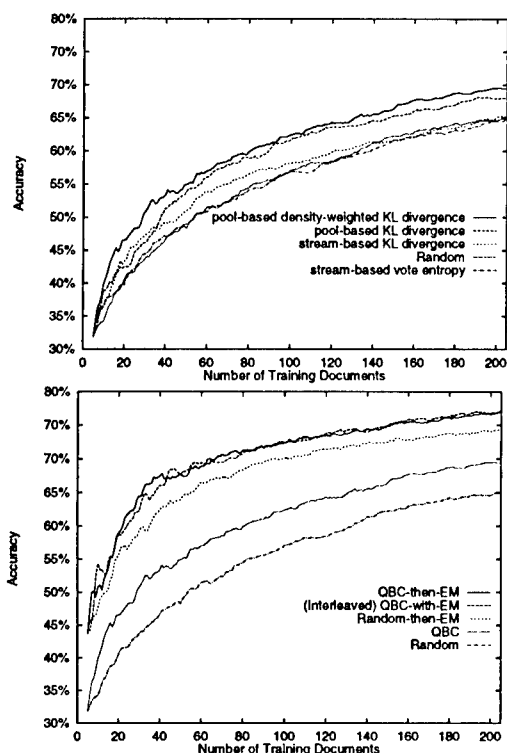


Figure 1: On the top, a comparison of disagreement metrics and selection strategies for QBC shows that density-weighted pool-based KL sampling does better than other metrics. On the bottom, combinations of QBC and EM outperform stand-alone QBC or EM. In these cases, QBC uses density-weighted pool-based KL sampling. Note that the order of the legend matches the order of the curves and that, for resolution, the vertical axes do not range from 0 to 100.

Surprisingly, stream-based vote entropy does slightly worse than random, needing 61 documents for the 51% threshold. Density-weighted pool-based sampling with a KL metric is statistically significantly better than each of the other methods ($p < 0.005$ for each pairing). It is interesting to note that the first several documents selected by this approach are usually FAQs for the various newsgroups. Thus, using a pool of unlabeled data can notably improve active learning.

In contrast to earlier work on part-of-speech tagging [Dagan & Engelson 1995], vote entropy does not perform well on document classification. In our experience, vote entropy tends to select outliers—documents that are short or unusual. We conjecture that this occurs because short documents and documents consisting of infrequently occurring words are the documents that most easily have their classifications changed by perturbations in the classifier parameters. In these situations, classification variance is high, but the dif-

ference in magnitude between the classification score of the winner and the losers is small. For vote entropy, these are prime selection candidates, but KL divergence accounts for the magnitude of the differences, and thus helps measure the confidence in the disagreement. Furthermore, incorporating density-weighting biases selection towards longer documents, since these documents have word distributions that are more representative of the corpus, and thus are considered “more dense.” It is generally better to label long rather than short documents because, for the same labeling effort, a long document provides information about more words. Dagan and Engelson’s domain, part-of-speech tagging, does not have varying length examples; document classification does.

Now consider the addition of EM to the learning scheme. Our EM baseline post-processes random selection with runs of EM (Random-then-EM). The most straightforward method of combining EM and active learning is to run EM after active learning completes (QBC-then-EM). We also interleave EM and active learning, by running EM on each committee member (QBC-with-EM). This also includes a post-processing run of EM. In QBC, documents are selected by density-weighted pool-based KL, as the previous experiment indicated was best. Random selection (Random) and QBC without EM (QBC) are repeated from the previous experiment for comparison.

The bottom graph of Figure 1 shows the results of combining EM and active learning. Starting with the 30 labeling mark again, QBC-then-EM is impressive, reaching 64% accuracy. Interleaved QBC-with-EM lags only slightly, requiring 32 labeled documents for 64% accuracy. Random-then-EM is the next best performer, needing 51 labeled documents. QBC, without EM, takes 118 labeled documents, and our baseline, Random, takes 179 labeled documents to reach 64% accuracy. QBC-then-EM and QBC-with-EM are not statistically significantly different ($p = 0.71$ N.S.); these two are each statistically significantly better than each of the other methods at this threshold ($p < 0.05$).

These results indicate that the combination of EM and active learning provides a large benefit. However, QBC interleaved with EM does not perform better than QBC followed by EM—not what we were expecting. We hypothesize that while the interleaved method tends to label documents that EM cannot reliably label on its own, these documents do not provide the most beneficial starting point for EM’s hill-climbing. In ongoing work we are examining this more closely and investigating improvements.

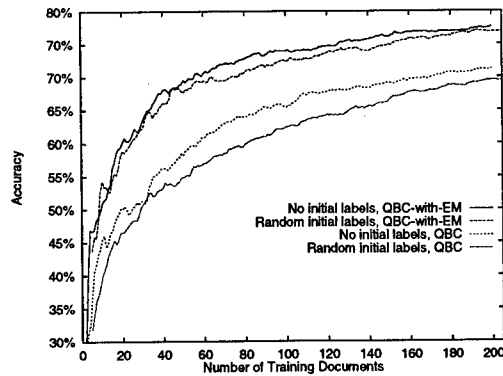


Figure 2: A comparison of random initial labeling and no initial labeling when documents are selected with density-weighted pool-based sampling. Note that no initial labeling tends to dominate the random initial labeling cases.

Another application of the unlabeled pool to guiding active learning is the selection of the initial labeled examples. Several previous implementations [Dagan & Engelson 1995; Lewis & Gale 1994; Lewis 1995] suppose that the learner is provided with a collection of labeled examples at the beginning of active learning. However, obtaining labels for these initial examples (and making sure we have examples from each class) can itself be an expensive proposition. Alternatively, our method can begin without any labeled documents, sampling from the Dirichlet distribution and selecting with density-weighted metrics as usual. Figure 2 shows results from experiments that begin with zero labeled documents, and use the structure of the unlabeled data pool to select initial labeling requests. Interestingly, this approach is not only more convenient for many real-world tasks, but also performs better because, even without any labeled documents, it can still select documents in dense regions. With 70 labeled documents, QBC initialized with one (randomly selected) document per class attains an average of 59% accuracy, while QBC initialized with none (relying on density-weighted KL divergence to select all 70) attains an average of 63%. Performance also increased with EM; QBC-with-EM rises from 69% to 72% when active learning begins with zero labeled documents. Each of these differences is statistically significant ($p < 0.005$). Both with and without EM, this method successfully finds labeling requests to cover all classes. As before, the first requests tend to be FAQs or similar, long, informative documents.

In comparison to previous active learning studies in text classification domains [Lewis & Gale 1994; Liere & Tadepalli 1997], the magnitude of our classification accuracy increase is relatively modest. Both

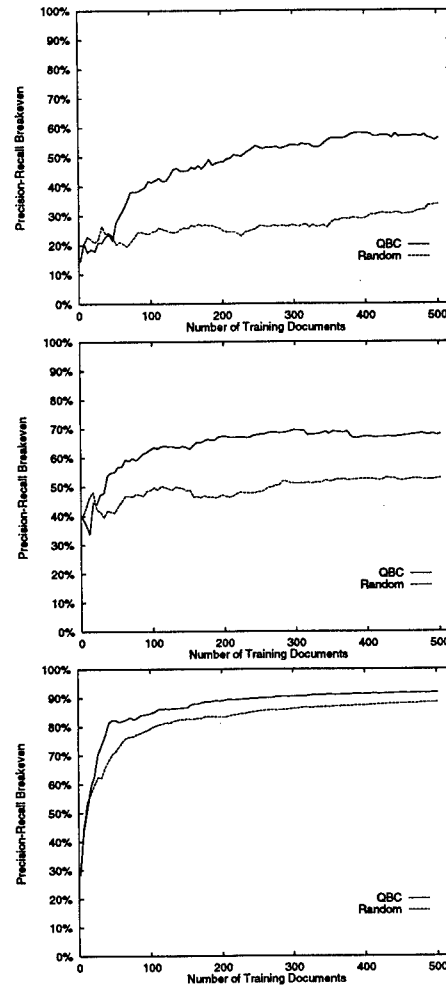


Figure 3: Active learning results on three categories of the Reuters data, corn, trade, and acq, respectively from the top and in increasing order of frequency. Note that active learning with committees outperforms random selection and that the magnitude of improvement is larger for more infrequent classes.

of these previous studies consider binary classifiers with skewed distributions in which the positive class has a very small prior probability. With a very infrequent positive class, random selection should perform extremely poorly because nearly all documents selected for labeling will be from the negative class. In tasks where the class priors are more even, random selection should perform much better—making the improvement of active learning less dramatic. With an eye towards testing this hypothesis, we perform a subset of our previous experiments on the Reuters data set, which has these skewed priors. We compare Random against unweighted pool-based sampling (QBC) with the KL disagreement metric.

Figure 3 shows results for three of the ten binary classification tasks. The frequencies of the positive classes are 0.018, 0.038 and 0.184 for the corn (top), trade (middle) and acq (bottom) graphs, respectively. The class frequency and active learning results are representative of the spectrum of the ten classes. In all cases, active learning classification is more accurate than Random. After 252 labelings, improvements of accuracy over random are from 27% to 53% for corn, 48% to 68% for trade, and 85% to 90% for acq. The distinct trend across all ten categories is that the less frequently occurring positive classes show larger improvements with active learning. Thus, we conclude that our earlier accuracy improvements are good, given that with unskewed class priors, Random selection provides a relatively strong performance baseline.

7 Conclusions

This paper demonstrates that by leveraging a large pool of unlabeled documents in two ways—using EM and density-weighted pool-based sampling—we can strongly reduce the need for labeled examples. In future work, we will explore the use of a more direct approximation of the expected reduction in classification variance across the distribution. We will consider the effect of the poor probability estimates given by naive Bayes by exploring other classifiers that give more realistic probability estimates. We will also further investigate ways of interleaving active learning and EM to achieve a more than additive benefit.

Acknowledgments

We are grateful to Larry Wasserman for help on theoretical aspects of this work. We thank Doug Baker for help formatting the Reuters data set. Two anonymous reviewers provided very helpful comments. This research was supported in part by the Darpa HPKB program under contract F30602-97-1-0215.

References

- Cohn, D.; Ghahramani, Z.; and Jordan, M. 1996. Active learning with statistical models. *Journal of Artificial Intelligence Research* 4:129–145.
- Cohn, D. 1994. Neural network exploration using optimal experiment design. In *NIPS 6*.
- Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 1998. Learning to extract symbolic knowledge from the World Wide Web. In *AAAI-98*.
- Dagan, I., and Engelson, S. 1995. Committee-based sampling for training probabilistic classifiers. In *ICML-95*.
- Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 39:1–38.
- Domingos, P., and Pazzani, M. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29:103–130.
- Freund, Y.; Seung, H.; Shamir, E.; and Tishby, N. 1997. Selective sampling using the query by committee algorithm. *Machine Learning* 28:133–168.
- Friedman, J. H. 1997. On bias, variance, 0/1 - loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery* 1:55–77.
- Ghahramani, Z., and Jordan, M. 1994. Supervised learning from incomplete data via an EM approach. In *NIPS 6*.
- Joachims, T. 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *ICML-97*.
- Joachims, T. 1998. Text categorization with Support Vector Machines: Learning with many relevant features. In *ECML-98*.
- Lewis, D., and Gale, W. 1994. A sequential algorithm for training text classifiers. In *Proceedings of ACM SIGIR*.
- Lewis, D., and Ringuette, M. 1994. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, 81–93.
- Lewis, D. D. 1995. A sequential algorithm for training text classifiers: Corrigendum and additional data. *SIGIR Forum* 29(2):13–19.
- Liere, R., and Tadepalli, P. 1997. Active learning with committees for text categorization. In *AAAI-97*.
- McCallum, A., and Nigam, K. 1998. A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*. <http://www.cs.cmu.edu/~mccallum>.
- Miller, D. J., and Uyar, H. S. 1997. A mixture of experts classifier with learning based on both labelled and unlabelled data. In *NIPS 9*.
- Nigam, K.; McCallum, A.; Thrun, S.; and Mitchell, T. 1998. Learning to classify text from labeled and unlabeled documents. In *AAAI-98*.
- Pereira, F.; Tishby, N.; and Lee, L. 1993. Distributional clustering of English words. In *Proceedings of the 31st ACL*.
- Shahshahani, B., and Landgrebe, D. 1994. The effect of unlabeled samples in reducing the small sample size problem and mitigating the Hughes phenomenon. *IEEE Trans. on Geoscience and Remote Sensing* 32(5):1087–1095.

Improving Text Classification by Shrinkage in a Hierarchy of Classes

Andrew McCallum*[†]
mccallum@justresearch.com

Ronald Rosenfeld[†]
roni@cs.cmu.edu

Tom Mitchell[†]
mitchell+@cs.cmu.edu

Andrew Y. Ng[‡]
ayn@ai.mit.edu

*Just Research
4616 Henry Street
Pittsburgh, PA 15213

[†]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[‡]MIT AI Lab
545 Technology Square
Cambridge, MA 02139

Abstract

When documents are organized in a large number of topic categories, the categories are often arranged in a hierarchy. The U.S. patent database and Yahoo are two examples.

This paper shows that the accuracy of a naive Bayes text classifier can be significantly improved by taking advantage of a hierarchy of classes. We adopt an established statistical technique called *shrinkage* that smoothes parameter estimates of a data-sparse child with its parent in order to obtain more robust parameter estimates. The approach is also employed in *deleted interpolation*, a technique for smoothing *n*-grams in language modeling for speech recognition.

Our method scales well to large data sets, with numerous categories in large hierarchies. Experimental results on three real-world data sets from UseNet, Yahoo, and corporate web pages show improved performance, with a reduction in error up to 29% over the traditional flat classifier.

1 Introduction

As the dramatic expansion of the World Wide Web continues, and the amount of on-line text grows, the development of methods for automatically categorizing this text becomes more important. A variety of recent work has demonstrated the success of statistical approaches for learning to classify text documents [Joachims 1997; Koller & Sahami 1997; Yang & Pederson 1997; Nigam *et al.* 1998]. These approaches, such as TFIDF [Salton 1991] and naive Bayes [Lewis & Ringuette 1994; McCallum & Nigam

1998], typically represent documents as vectors of words, and learn by gathering statistics from the observed frequencies of these words within documents belonging to the different classes. Because they rely on these learned word statistics, these approaches are data-intensive: they often require large numbers of hand-labeled training documents per class to achieve high classification accuracy.

This paper considers the question of how to scale up these statistical learning algorithms to tasks with a large number of classes and sparse training data per class. When humans organize extensive data sets into fine-grained categories, topic hierarchies are often employed to make the large collection of categories more manageable. Yahoo, the U.S. patent database, MEDLINE and the Dewey Decimal System are all examples of such hierarchies.

We present a technique that leverages these commonly-available topic hierarchies in order to significantly improve classification accuracy, especially when the hierarchy is large and the training data for each class is sparse. We also present a method for exponentially reducing the amount of computation necessary for classification, while sacrificing only a small amount of accuracy.

Our approach applies a well-understood technique from Statistics called *shrinkage* that provides improved estimates of parameters that would otherwise be uncertain due to limited amounts of training data [Stein 1955; James & Stein 1961]. The technique exploits a hierarchy by “shrinking” parameter estimates in data-sparse children toward the estimates of the data-rich ancestors in ways that are provably optimal under the appropriate conditions. We employ a simple form of shrinkage that creates new parameter estimates in a child by a linear interpolation of all hierarchy nodes from the child to the root. The interpolation weights

are learned by a form of Expectation Maximization [Dempster, Laird, & Rubin 1977]. This form of shrinkage is also applied in *deleted interpolation*, a technique for smoothing n -grams in language modeling for speech recognition [Jelinek & Mercer 1980].

Note that our approach to text classification in a hierarchy is quite different than work by Koller and Sahami [Koller & Sahami 1997]. Their *Pachinko Machine* employs the hierarchy by learning separate classifiers at each internal node of the tree, and then labeling a document by using these classifiers to greedily select sub-branches until it reaches a leaf. Their approach is shown to be helpful when documents are represented using a small subset (< 100 words) of the available vocabulary, and a different subset of the vocabulary is selected at each node of the hierarchy. However, their approach did not show improvement with larger vocabularies, and in many domains (including the domains studied in this paper) it has been established that large vocabulary sizes often perform best [Joachims 1997; Nigam *et al.* 1998; McCallum & Nigam 1998].

Somewhat surprisingly, it can be shown that a probabilistic form of Pachinko Machine, when trained using maximum likelihood estimates and a constant vocabulary, is equivalent to the simple non-hierarchical classifier [Mitchell 1998]. At each node in the hierarchy this non-deterministic version of the Pachinko Machine assigns each document probabilistically to all of its descendants, whereas the deterministic Pachinko Machine proposed by Koller and Sahami assigns each document to its single most probable descendant.

The remainder of this paper is structured as follows: we explain our probabilistic approach to text classification, and present the use of shrinkage in this context. Then we show experimental results on three real-world data sets, present related work, and close with a discussion of future work.

2 Probabilistic Framework

We approach the task of text classification in a Bayesian learning framework. We assume that the text data was generated by a parametric model, and use training data to calculate estimates of the model parameters. Then, equipped with these estimates, we classify new test documents by using Bayes rule to turn the generative model around and calculate the posterior probability that a class would have generated the test document in question. Classification then becomes a simple matter of selecting the most probable

class given the document's words.

We assume that the data is generated by a mixture model, (parameterized by θ), with a one-to-one correspondence between mixture model components and (the observed) classes, $c_j \in \{C\}$. This specifies that a document, d_i , is created by (1) selecting a class, c_j , according to the class priors, $P(c_j|\theta)$, then (2) having the corresponding mixture component generate a document according to its own parameters, with distribution $P(d_i|c_j; \theta)$. The marginal probability of generating document d_i is thus a sum of total probability over all mixture components:

$$P(d_i|\theta) = \sum_{j=1}^{|C|} P(c_j|\theta)P(d_i|c_j; \theta). \quad (1)$$

A document is comprised of an ordered sequence of word events, drawn from a vocabulary V . We make the naive Bayes assumption: that the probability of each word event in a document is independent of the word's context given the class, and furthermore independent of its position in the document. Thus, each document d_i is drawn from a multinomial distribution with as many independent trials as the number of words in d_i . We also assume that document lengths, $|d_i|$, are independent of class. We write $w_{d_i,k}$ for the word in position k of document d_i , where the subscript of w (in this case $d_{i,k}$) indicates an index into the vocabulary. Then the probability of a document given its class is:

$$P(d_i|c_j; \theta) = P(|d_i|) \prod_{k=1}^{|d_i|} P(w_{d_i,k} | c_j; \theta). \quad (2)$$

Given the assumption about one-to-one correspondence between mixture model components and classes, the naive Bayes assumption, and the position independence assumption, the mixture model is composed of disjoint sets of parameters, θ_j , for each class c_j . This parameter set for each class, θ_j , is composed of probabilities for each word, w_t , such that $\theta_{jt} \equiv P(w_t|c_j; \theta)$ and $\sum_{t=1}^{|V|} \theta_{jt} = 1$. The only other parameters in the model are the class prior probabilities, written $\theta_{0j} \equiv P(c_j|\theta)$.

Given a set of labeled training documents, \mathcal{D} , we can calculate estimates for the parameters of the model that generated the documents. These estimates consist of straightforward counting of events, supplemented by standard Laplace 'smoothing' that primes each estimate with a count of one to avoid probabilities of zero. We define $N(w_t, d_i)$ to be the count of the number of times word w_t occurs in document d_i , and

define $P(c_j|d_i) \in \{0,1\}$, as given by the document's class label. Then, the estimate of the probability of word w_t in class c_j is

$$\hat{\theta}_{jt} \equiv P(w_t|c_j; \hat{\theta}) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} N(w_t, d_i) P(c_j|d_i)}{|V| + \sum_{s=1}^{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{D}|} N(w_s, d_i) P(c_j|d_i)} \quad (3)$$

The class prior parameters are set by the maximum likelihood estimate:

$$\hat{\theta}_{0j} \equiv P(c_j|\hat{\theta}) = \sum_{i=1}^{|\mathcal{D}|} P(c_j|d_i) / |\mathcal{D}|. \quad (4)$$

Given estimates of these parameters calculated from the training documents, classification can be performed on test documents by calculating the posterior probability of each class given the words observed in the test document, and selecting the class with the highest probability. We formulate this by first applying Bayes rule, and then substituting for $P(d_i|c_j; \theta)$ and $P(d_i|\theta)$ using Equations 1 and 2.

$$\begin{aligned} P(c_j|d_i; \hat{\theta}) &= \frac{P(c_j|\hat{\theta}) P(d_i|c_j; \hat{\theta})}{P(d_i|\hat{\theta})} \\ &= \frac{P(c_j|\hat{\theta}) \prod_{k=1}^{|d_i|} P(w_{d_{ik}}|c_j; \hat{\theta})}{\sum_{r=1}^{|\mathcal{C}|} P(c_r|\hat{\theta}) \prod_{k=1}^{|d_i|} P(w_{d_{ik}}|c_r; \hat{\theta})} \end{aligned} \quad (5)$$

Despite the fact that the mixture model and word independence assumptions are strongly violated with real-world data, naive Bayes performs text classification very well. Friedman and Domingos and Pazzani discuss why the violation of the word independence assumption sometimes does little damage to classification accuracy [Friedman 1997; Domingos & Pazzani 1997].

3 Hierarchical Classification

This section presents a method of improving our estimates of the model parameters by taking advantage of the hierarchy. We first briefly describe *shrinkage* in a general sense, then discuss its application to text classification in a hierarchy, and the mechanics of our algorithm.

Background on Shrinkage

We wish to estimate parameters $\theta_1, \dots, \theta_{|C|}$, (i.e. each class's probability distribution over words). The estimates $\hat{\theta}_j$ of θ_j can often be improved by shrinking

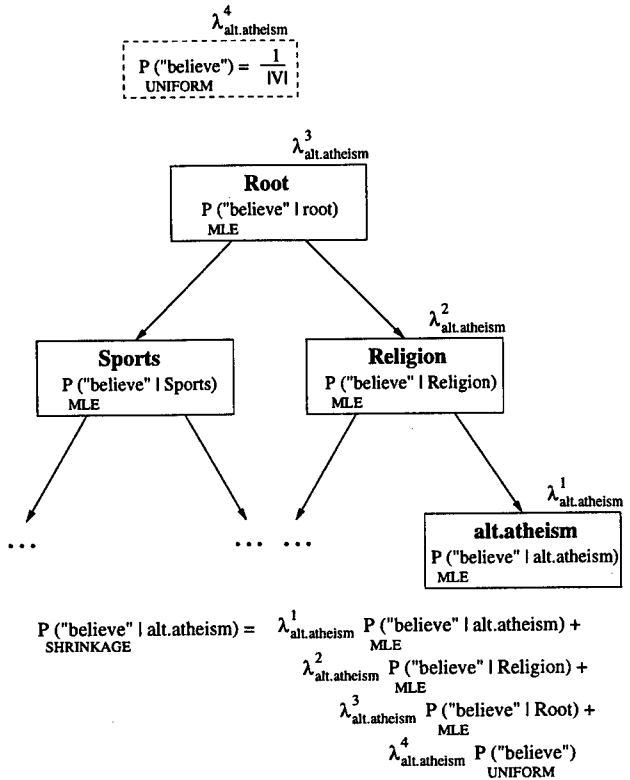


Figure 1: The new, shrinkage-based estimate of the probability of a word (e.g. "believe") given a UseNet class (e.g. alt.atheism) is a weighted sum of the maximum-likelihood estimates from the leaf to the root, and beyond the root to the uniform distribution over words.

each of them towards some common value. See Carlin and Louis [1996] for a recent summary of shrinkage. There are two justifications for shrinkage. First, if the quantities $\theta_1, \dots, \theta_{|C|}$ are thought to be similar, then they can be regarded as draws from a common distribution. In this case, the shrinkage estimator is just the Bayes estimate. More surprisingly, even if the quantities are completely unrelated, and even if the data upon which each estimator is based are independent of each other, shrinkage estimators still reduce the risk of the estimators. This is a deep and counterintuitive fact discovered by Stein [1955] and James and Stein [1961].

Shrinkage for Text Classification

We use shrinkage to better estimate the probability θ_{jt} of word w_t given class c_j . For each node in our tree we construct a maximum likelihood (ML) estimate based on the data associated with that node (Equation 3 without the Laplace smoothing). An improved estimate for each leaf node is then derived by "shrinking" its ML estimate towards the ML estimates of all its an-

cestors, namely those estimates found along the path from that leaf to the root. Figure 1 illustrates this process. In statistical language modeling terms, we build a unigram model for each node in the tree, and smooth each leaf model by linearly interpolating it with all the models found along the path to the root.

The estimates along a path from the leaf to the root represent a tradeoff between specificity and reliability. The estimate at the leaf is the most specific (most pertinent, least biased), since it is based on data from that topic alone. However it is also the least reliable, since it is based on the smallest sample of data. The estimator at the root is the most reliable, but the least specific.

Since even the root contains a finite amount of data, it may estimate some rare words unreliably. We therefore extend the tree by adding, beyond the root, the uniform estimate. Thanks to the latter, we no longer need to smooth the individual ML estimates with the Laplacean prior.

To ensure that the ML estimates along a given path are independent, we subtract each child's data from its parent's before calculating the parent's ML estimate. Thus the latter estimate is based on data that belongs to all the siblings of said child, but not to the child itself. Note that in this way, for any path from leaf to root, every datum in the tree is used in *exactly one* of the ML estimates, providing both independence among the estimates and efficient use of the training data.

Determining Mixture Weights

Given a set of ML estimates along the path from a leaf to the root (and beyond it, to the uniform estimate), how do we decide on the weights for interpolating (mixing) them? Let $\{\hat{\theta}_j^1, \hat{\theta}_j^2, \dots, \hat{\theta}_j^k\}$ be k such estimates, where $\hat{\theta}_j^1 = \hat{\theta}_j$ is the estimate at the leaf, and $\hat{\theta}_j^k$ is the uniform estimate ($\hat{\theta}_{jt}^k = 1/|V|$ for all words w_t), and $k-2$ is the depth of class c_j in the tree. The interpolation weights among the ancestors of class c_j are written $\{\lambda_j^1, \lambda_j^2, \dots, \lambda_j^k\}$, where $\sum_{i=1}^k \lambda_j^i = 1$.

We write $\tilde{\theta}_j$ for the new estimate of the class-conditioned word probabilities based on shrinkage. The new estimate for the probability of word w_t given class c_j is

$$\tilde{\theta}_{jt} = P(w_t|c_j; \tilde{\theta}_j) = \lambda_j^1 \hat{\theta}_{jt}^1 + \lambda_j^2 \hat{\theta}_{jt}^2 + \dots + \lambda_j^k \hat{\theta}_{jt}^k. \quad (6)$$

We derive empirically optimal weights, λ_j^i , between the ancestors of c_j , by finding the weights that maxi-

mize the likelihood of some hitherto unseen "held-out" data. We use the fact that the likelihood of data according to the mixture model is a convex function of the weights (this falls out of Jensen's inequality), and thus attains a single, global maximum. We find that maximum for each leaf class, c_j , using the following iterative procedure:

Initialize: Set the λ_j 's to some initial values, say $\lambda_j^i = \frac{1}{k}$ (any normalized non-zero initial values will do).

Iterate:

(1) Calculate the degree to which each estimate predicts the words w_t in the held-out set, \mathcal{H}_j , from class c_j :

$$\begin{aligned} \beta_j^i &= \sum_{w_t \in \mathcal{H}_j} P(\hat{\theta}_j^i \text{ was used to generate } w_t) \\ &= \sum_{w_t \in \mathcal{H}_j} \frac{\lambda_j^i \hat{\theta}_{jt}^i}{\sum_m \lambda_j^m \hat{\theta}_{jt}^m} \end{aligned} \quad (7)$$

(2) Derive new (and guaranteed improved) weights by normalizing the β 's:

$$\lambda_j^i = \frac{\beta_j^i}{\sum_m \beta_j^m} \quad (8)$$

Terminate: Upon convergence of the likelihood function (usually achieved within a dozen or so iterations).

This algorithm can be viewed as a particularly simple form of EM [Dempster, Laird, & Rubin 1977], where each datum is assumed to have been generated by first choosing one of the tree nodes in the path to the root, say $\hat{\theta}_j^i$ (with probability λ_j^i), then using that estimate to generate that datum. EM then maximizes the total likelihood when the choices of estimates made for the various data are unknown. The first step in the iterative part is thus the "E" step, and the second one is the "M" step.

While conceptually simple, this method makes inefficient use of the available training data by carving off some of it to be used as a held-out set. To overcome this problem, we modify the algorithm as follows: all the available data is used both to construct the ML estimates and to optimize the weights. However, as each document is used in the above algorithm, the ML estimates are modified to exclude its data, so as to make

# training documents	Class	Mixture Weights			
		child	parent	g'parent	uniform
235	root/politics/talk.politics.guns	0.368	0.092	0.017	0.522
	root/politics/talk.politics.mideast	0.256	0.132	0.001	0.611
	root/politics/talk.politics.misc	0.197	0.213	0.026	0.564
	root/religion/alt.atheism	0.235	0.158	0.022	0.585
	root/religion/soc.religion.christian	0.181	0.189	0.052	0.578
	root/religion/talk.religion.misc	0.104	0.255	0.028	0.613
7497	root/politics/talk.politics.guns	0.801	0.089	0.048	0.061
	root/politics/talk.politics.mideast	0.859	0.061	0.010	0.071
	root/politics/talk.politics.misc	0.762	0.126	0.043	0.068
	root/religion/alt.atheism	0.766	0.174	0.043	0.018
	root/religion/soc.religion.christian	0.837	0.098	0.041	0.024
	root/religion/talk.religion.misc	0.663	0.226	0.049	0.062

Table 1: Mixture weights learned by EM for some nodes in the UseNet class hierarchy described in section 4. Notice that when training data is sparse (top half of table), classes mix more strongly with their parents than when data is plentiful. Notice also that more 'generic' classes mix more strongly with their parents, *e.g.* talk.politics.misc's weight on its parent is higher than is talk.politics.guns's).

them independent of it. This method is very similar to the "leave-one-out" cross-validation commonly used in statistical estimation.

This technique of finding the optimal weights is routinely used in statistical language modeling to interpolate together different models (such as trigram, bigram, unigram and uniform), where it is known as "deleted interpolation" [Jelinek & Mercer 1980]. It was similarly used to interpolate estimates from nodes along a tree path in [Bahl *et al.* 1989]. This cross-validation approach to setting the mixture weights is not exactly the same style of shrinkage as Stein [1955] and James and Stein [1961], but is similar in spirit. In future work we will compare the different styles of shrinkage.

Table 1 shows a subset of the mixture weights learned by EM for a hierarchy based on UseNet articles.

4 Experimental Results

This section provides empirical evidence that shrinkage reduces text classification error by up to 29%. We also show that shrinkage helps most when training data is sparse and the number of classes is large. Finally, we demonstrate that dynamically pruning the tree can exponentially reduce computation time, at minimal loss of accuracy. Experiments are based on three different real-world data sets, one consisting of UseNet articles, and two of web pages.¹ All the results are averages of ten cross-validation trials.

¹All three data sets are available on-line. See <http://www.cs.cmu.edu/~textlearning>.

The Industry Sector hierarchy, made available by *Market Guide Inc.* (www.marketguide.com), consists of company web pages classified in a hierarchy of industry sectors. Using all classes at depth two results in 6440 web pages partitioned into 71 classes. In tokenizing the data we skip all MIME headers and HTML tags, use a stoplist, but do not stem. After removing tokens that occur only once, the corpus contains 1.2 million words, with a vocabulary of size 29964.

The Newsgroups data set, collected by Ken Lang, contains about 20,000 articles evenly divided among 20 UseNet discussion groups [Joachims 1997]. Several of the topic classes are quite confusable: five of them are about computers; three discuss religion. From this data set, we build a two-level hierarchy from the 15 classes that fit into the following top level categories: vehicles, computers, politics, religion and sports. We tokenize the data in the same way as above. The resulting data set, after removing words that occur only once, contains 1.7 million words, and a vocabulary size of 52309.

We gathered the entirety of the Yahoo 'Science' hierarchy in July 1997. The web pages pointed to by Yahoo are divided into 264 disjoint classes containing 14831 pages as result of descending to deeper nodes of Yahoo's hierarchy until each class contains less than 200 documents, and then removing classes with fewer than 20 documents. After tokenizing as above and removing stopwords and words that occur only once, the corpus contains 3.0 million words, with a vocabulary size of 76624.

Feature selection, when used, is performed by select-

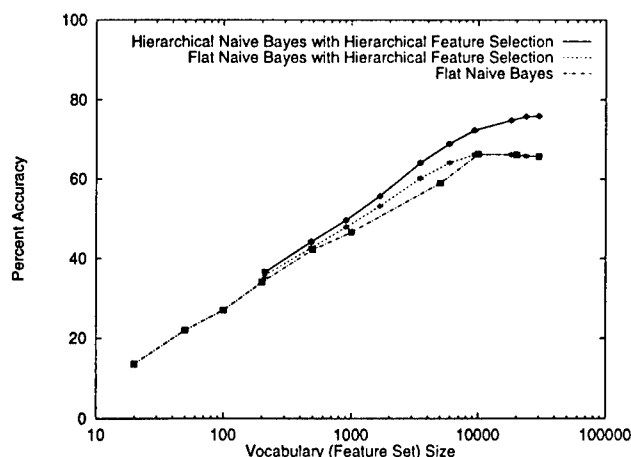


Figure 2: Classification accuracy on the Industry Sector data set with varying vocabulary size in the horizontal axis. The tiny vertical bars at each data point indicate standard error. Performance is best with the full vocabulary, where shrinkage reduces error by almost one-third.

ing the words that have highest mutual information with the class variable. A previous study found this method to be the best for text among several common methods [Yang & Pederson 1997]. In addition to selecting features by the traditional, flat use of mutual information, we also use the hierarchy for feature selection. *Hierarchical feature selection* selects equal numbers of top words by mutual information at each internal node of the tree, using the node's immediate children as the classes. This corresponds to Koller and Sahami's hierarchical feature selection with zero dependencies [Koller & Sahami 1997], except that we define the total vocabulary to be the union of all the vocabularies chosen by the internal nodes. The union is necessary so that the models we will mix share the same event space.

Hierarchical classification improves accuracy

Figure 2 shows classification accuracy on the Industry Sector data set with 50-50 train-test splits while varying vocabulary size. No partial credit is given for classification into neighbors of the true class.

First, note that larger vocabulary sizes generally perform better; this is consistent with previous results of naive Bayes on several other data sets [Joachims 1997; Nigam *et al.* 1998; McCallum & Nigam 1998]. Second, note that Hierarchical Feature Selection somewhat improves the performance of flat naive Bayes in the mid-range of feature selection—at about 5000 words, traditional, flat feature selection obtains 59% accuracy, while hierarchical feature selection reaches

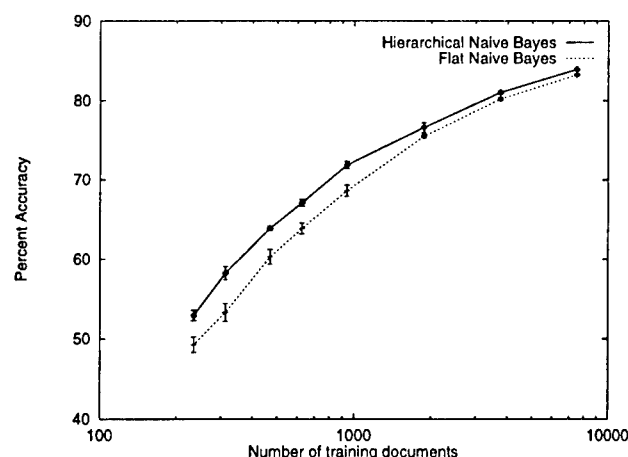


Figure 3: Classification accuracy on the Newsgroups data set with varying amounts of training data. The vertical axis is zoomed for magnification of the error bars. Overall, hierarchical modeling provides less improvement than it does in the Industry Sector data set because the hierarchy is much smaller. Notice, however, that, as expected, shrinkage helps more when there is less training data.

64%. Third, and most importantly, observe that shrinkage improves classification accuracy across the board, making the largest improvement at the full, unpruned vocabulary size, where it achieves 76% accuracy. In comparison, the flat classifier reaches its best performance of 66% at about 10000 words. This difference represents a 29% reduction in classification error. We maintain that low-frequency words contribute significantly to correct classifications, and that shrinkage helps reduce variance of the estimates in the larger parameter space that results from the larger vocabulary.²

Shrinkage helps more when training data is sparse.

Figure 3 shows accuracy on the Newsgroups data set with the full vocabulary, while varying amount of training data. Our experiments indicate that accuracy in this domain is highest with no feature selection, (*i.e.* using the full vocabulary), for both flat and hierarchical classifiers, even with small amounts of training data.

It is interesting to see that hierarchical modeling provides less improvement on this data set than it does in the Industry Sector corpus. We expect that this is

²Large vocabularies need not be a computational concern. In our experiments, with the largest vocabulary, it takes only 216 seconds to classify 3220 Industry Sector documents and write the results to disk. In comparison, the smallest vocabulary takes 208 seconds—a difference of 0.002 seconds per document on average.

due to the significantly reduced branch-out factor in this smaller hierarchy. Unlike the Industry Sector hierarchy, in which the mean number of siblings is six, here the mean number of siblings is three. Thus each child has fewer siblings and less data from which to "borrow strength."

The second expected result, exhibited in Figure 3, is that shrinkage provides more improvement when the amount of training data is small, and that shrinkage reduces variance in the classifications; (notice larger error bars on the 'flat classification' curve). If each class had an infinite amount of training data, accurate parameter estimates could be obtained for each class independently; however, when training data is sparse, estimates are improved by using shrinkage to smooth a class's parameters with its ancestors.

The two findings that (1) shrinkage allows the use of helpful large vocabulary sizes, and (2) shrinkage improves performance more when training data is sparse, are both confirmed by our experiments with the Yahoo data set. Figure 4 shows classification accuracy on the Science hierarchy as a function of vocabulary size, again, with no partial credit for near misses. Flat naive Bayes reaches its highest accuracy of 36.4% at a relatively small vocabulary size of 1449. Hierarchical classification always performs better than flat, but attains its best accuracy of 39.5% at a larger vocabulary size of 13311. The improvement in accuracy is not as dramatic here as with the Industry Sector data set, perhaps because the Yahoo set is more noisy (being gathered automatically rather than by hand, and containing many documents that are simply timeout messages or pointers to moved pages), and because Yahoo has many classes with overlapping or closely neighboring definitions.³ However, it is interesting to note that among those classes with small quantities of training data, shrinkage improves performance more strongly. Among those 151 classes with 50 documents or less, shrinkage improves accuracy by 8%, from 29% to 37%. Among those 50 classes containing more than 100 documents, shrinkage does not improve accuracy, both obtaining about 45%.

This result indicates that shrinkage would be all the more important if we attempted to classify documents into Yahoo's deepest leaf categories instead of into the somewhat coalesced and pruned version that is used

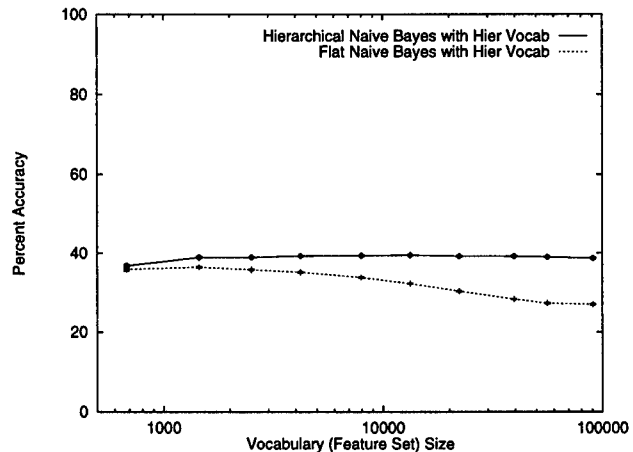


Figure 4: Classification accuracy on the Yahoo Science data set with varying vocabulary sizes. Tiny vertical bars at each data point indicate standard error. The large number of classes and noisy data make this task difficult.

here and is defined at the beginning of this section. However, this would result in thousands of classes—quite a computational burden. Next we describe how the hierarchy itself can be used to ease this burden.

Pruning the tree for increased computational efficiency

In addition to improving accuracy, the class hierarchy can also be leveraged to improve computational efficiency. The classifier can avoid calculating $P(c_j|d_i)$ for a majority of the classes (leaves of the tree) by pruning the tree dynamically during the classification of each document. Like the *Pachinko Machine* [Koller & Sahami 1997] we can classify the document at internal nodes of the tree, and choose only to calculate probabilities for classes underneath the branches selected by these higher-level, coarse-grained classifiers. Note, however, that when we do this, each "pruning classification" at the interior of the tree is an opportunity for error, and the deeper the hierarchy the more the opportunities for error will compound.

As expected, our experimental results show that performing this pruning does indeed reduce classification accuracy. However, one may be willing to accept this reduction in exchange for the exponential reduction in the amount of computation necessary for classification. On the Industry Sector data set, averaged over ten runs, pruning that removes from consideration all but a single branch at each interior node reaches 70.0% accuracy, more than 5% points lower than without pruning. However, unlike the *Pachinko Machine*, our paradigm allows for the comparison of classification scores from

³Using more complex Bayesian classifiers that capture more dependencies than naive Bayes may help this last problem. The larger number of parameters in these models will make training data even more sparse, and this suggests that the use of shrinkage would be all the more important.

leaves that do not share the same parent. Thus we can also prune less aggressively. Pruning that keeps two branches attains 74.3%. And pruning to three branches achieves 75.2%. This last result is only half a percent less than the 75.8% obtained by the full evaluation of the tree without pruning. The same approach could also be used for Yahoo.

5 Related Work

Shrinkage estimation is now considered standard methodology in Statistics. It is used routinely in a vast array of problems and its theoretical properties have been studied from both the Bayesian and frequentist points of view. A good discussion with ample references and examples is contained in [Carlin & Louis 1996]. Although MacKay and Peto [1994] do not use the term “shrinkage” in their paper, they apply this Bayesian style of shrinkage in their hierarchical Dirichlet model for n -grams.

Shrinkage in the cross-validation style was first used to derive a language model in [Jelinek & Mercer 1980], where it is known as *deleted interpolation*. Interpolation of language models along the path of a tree is described in [Bahl *et al.* 1989]. More recently, Seymore and Rosenfeld [1997] classified a speech recognizer’s output into multiple topics, then used an automatically derived “topic tree” to interpolate the models associated with appropriate nodes up that tree.

A variety of work in the Information Retrieval and Machine Learning communities has demonstrated the success of statistical approaches for learning to classify text documents. Naive Bayes has been used for text classification, and due to its probabilistic foundations, been applied in several extensions [Lewis & Ringuette 1994; Joachims 1997; Nigam *et al.* 1998].

An earlier approach to hierarchical document classification, the *Pachinko Machine*, has been proposed by Koller and Sahami [1997]. Their method differs significantly from shrinkage. The Pachinko Machine classifies documents at internal nodes of the tree, and greedily selects sub-branches until it reaches a leaf. Since classification errors at internal nodes compound, the accuracy at all the internal nodes must be very high in order for overall accuracy to be higher than a flat classifier (especially for deeper hierarchies). We experimented with schemes that allow a lower node to “reject” a document and send it back up the tree for re-classification, but did not find these to work well. Koller and Sahami present results with small vocabularies (less than 100 words); however, other

results in the literature indicate that large vocabulary sizes often have higher accuracy [Joachims 1997; Nigam *et al.* 1998]. A possible explanation for the discrepancy is that Koller and Sahami use a multi-variate Bernoulli model while we use a multinomial model [Sahami, Personal Communication]. In our experiments we have found multinomials to outperform Bernoullis [McCallum & Nigam 1998]. Our use of shrinkage has allowed us to more robustly keep large vocabulary sizes, which we believe are necessary for classifying large data sets with large numbers of diverse classes.

Another learning method that uses EM to set mixture weights among ancestors in a hierarchy is *Adaptive Mixtures of Probabilistic Transducers* [Singer 1997]. Each node in a hierarchy that represents a history-window is linearly mixed with its parent, which in turn, is mixed with its parent. The model is applied with success to noun phrase recognition.

Hofmann and Puzicha’s [1998] Hierarchical Asymmetric Clustering Model (HACM) performs unsupervised clustering with a mixture model in which EM is also used to set weights among the ancestors in a hierarchy.

6 Conclusions

This paper has examined the use of class hierarchies for improving text classification. As the amount of on-line text increases and the number of topic categories into which it is organized grows, hierarchies are becoming an increasingly prevalent way to make a collection of categories manageable. Thus, the need for good text classification algorithms that take advantage of these hierarchies becomes more important.

In this paper we demonstrate that shrinkage with a class hierarchy improves parameter estimation, and can reduce text classification error by up to 29%. Because shrinkage helps especially when there is sparse training data, shrinkage should be all the more beneficial as we scale up to larger, higher-resolution, deeper hierarchies with more classes that require larger vocabularies.

We also show that a class hierarchy can be used to exponentially reduce the amount of computation required to classify documents, and that we can do so without sacrificing significant classification accuracy.

In future work, we will investigate the use of shrinkage to learn more complex Bayesian models with less restrictive assumptions than naive Bayes. The improvements due to shrinkage should be increasingly strong

as we move to models that have more parameters, and thus sparser training data. We will also explore alternative methods of shrinkage, including the Bayesian methods in the style of James and Stein. We plan to work with a related approach that uses EM to cluster the data in a parent, and then allows the child to mix with the different clusters independently. In other ongoing work we are studying the advantages of using EM not only to set the mixture weights, but also redistribute individual words of training data among the nodes on the path from the leaf to the root.

Lastly, we plan to explore ways to learn the class hierarchy—investigating methods that specifically aim to increase classification accuracy. In early experiments, it appears that when the learner is not explicitly given a hierarchy, then even using the “trivial” hierarchy (each class being a leaf off the root) does better than the flat classifier, though not as well as when we are given a “non-trivial” hierarchy. Furthermore, using a “bad” or scrambled hierarchy also does better than the flat classifier—the mixture weights are set by EM to mimic the trivial hierarchy.

Acknowledgments

Larry Wasserman contributed many valuable discussions, pointers to the statistics literature, and comments for this paper. Kamal Nigam provided numerous helpful suggestions on earlier drafts. We thank *Market Guide, Inc.* for permission to use their Industry Sector hierarchy, and Mark Craven for gathering its data from the web. We thank *Yahoo!* for permission to use their data. This research was supported in part by the Darpa HPKB program under contract F30602-97-1-0215. A. Y. Ng is supported by the NSF under contract number ASC-92-17041.

References

- Bahl, L.; Brown, P.; deSouza, P.; and Mercer, R. 1989. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing* 37:1001–1008.
- Carlin, B., and Louis, T. 1996. *Bayes and Empirical Bayes Methods for Data Analysis*. Chapman and Hall.
- Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 39:1–38.
- Domingos, P., and Pazzani, M. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29:103–130.
- Friedman, J. H. 1997. On bias, variance, 0/1 - loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery* 1:55–77.
- Hofmann, T., and Puzicha, J. 1998. Statistical models for co-occurrence data. Technical report, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, MIT. AI Memo 1625, CBCL Memo 159.
- James, W., and Stein, C. 1961. Estimation with quadratic loss. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability 1*, 361–379. University of California Press.
- Jelinek, F., and Mercer, R. 1980. Interpolated estimation of Markov source parameters from sparse data. In Gelserma, S., and Kanal, L. N., eds., *Pattern Recognition in Practice*, 381–402.
- Joachims, T. 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *International Conference on Machine Learning (ICML)*.
- Koller, D., and Sahami, M. 1997. Hierarchically classifying documents using very few words. In *ICML-97: Proceedings of the Fourteenth International Conference on Machine Learning*, 170–178. Morgan Kaufmann.
- Lewis, D., and Ringuette, M. 1994. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, 81–93.
- MacKay, D., and Peto, L. B. 1994. A hierarchical dirichlet language model. *Natural Language Engineering* 1(1).
- McCallum, A., and Nigam, K. 1998. A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*. <http://www.cs.cmu.edu/~mccallum>.
- Mitchell, T. M. 1998. Conditions for the equivalence of hierarchical and flat Bayesian classifiers. <http://www.cs.cmu.edu/~tom/hierproof.ps>.
- Nigam, K.; McCallum, A.; Thrun, S.; and Mitchell, T. 1998. Learning to classify text from labeled and unlabeled documents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence, AAAI-98*.
- Salton, G. 1991. Developments in automatic text retrieval. *Science* 253:974–979.
- Seymore, K., and Rosenfeld, R. 1997. Using story topics for language model adaptation. In *Eurospeech*.
- Singer, Y. 1997. Adaptive mixtures of probabilistic transducers. *Neural Computation* 9(8).
- Stein, C. 1955. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability 1*, 197–206. University of California Press.
- Yang, Y., and Pederson, J. 1997. Feature selection in statistical learning of text categorization. In *ICML-97*, 412–420.

A Case Study in the Use of Theory Revision in Requirements Validation

T.L. McCluskey

School of Computing and Mathematics,
The University of Huddersfield,
HD1 3DH, UK
lee@zeus.hud.ac.uk.

M.M. West

School of Computing and Mathematics,
The University of Huddersfield,
HD1 3DH, UK
m.m.west@zeus.hud.ac.uk.

Abstract

Research emanating from Artificial Intelligence has throughout its history contributed to techniques and ideas in Software Engineering. We describe in this paper a case study showing the use of theory revision to the refinement of a formally specified requirements model. In a previous project we were contracted to create a precise model of the complex criteria governing the separation of aircraft profiles in Atlantic Airspace. During that work it became clear that the (automated) validation of the model was of the utmost importance, and in our current project we have used machine learning tools to provide extra support in bug identification, bug removal and maintenance of such a requirements model. In this paper we give an overview of the domain, identify a relevant learning bias which makes search for revisions tractable, and describe a systematic approach for the application of theory revision to such a model. We illustrate the approach with results of experiments where theory revision techniques have identified and removed errors, and induced a new part of the model.

Keywords Theory Revision, Machine Learning and Software Engineering, Requirements Model, Automated Validation.

1 INTRODUCTION

Promoting and maintaining the quality of requirements specifications has a vital role in the engineering of software. Some software projects, such as those

involving safety-critical elements, necessitate that precise, mathematical specifications of their requirements domains be constructed. Such 'requirements models' must be *validated* to satisfy certain major quality objectives such as accuracy, completeness, usability, and understandability, and during the model's lifetime it is likely to be incrementally updated, and will require re-validation. Validation and maintenance of realistic domain models is a very time consuming, expensive process where the role of support tools is vital. The process is best carried out using diverse techniques, and one of the most useful techniques is to *test* an animated form of the model. Even when an animated version is available, however, it is not easy to pinpoint the causes of bugs and subsequently provide the correct revision that eliminates them.

In this work we view a precise requirements model as an imperfect theory of the requirements domain that needs to undergo refinement to remove bugs or to reflect changes in the domain, and we formulate the problem as one of *theory revision*. The case study uses an air traffic control requirements model developed in a previous project called FAROAS (McCluskey et al. 1995). The model represents aircraft separation criteria and conflict prediction procedures relating to airspace over the North East Atlantic, and is recorded in the 'Formal Methods Europe Applications Database'¹. The model's 'conventional' support environment had been used for verification and validation of models written as a set of axioms in *many sorted first order logic* (Meinke and Tucker 1993) – here abbreviated to msl. During the current IMPRESS project we extended the environment to include machine learning tools which perform blame assignment, explanation-based generalisation and theory revision (TR). We show in this paper how we overcame the in-

¹web site <http://www.cs.tcd.ie/FME>

tractability problems in fielding TR by firstly focusing on likely faulty axioms sets using a blame assignment algorithm, then targeting for revision the ordering relations between values of *ordinal* sorts. We describe a method and a class of revision operator that has been successfully used to (a) find and remove bugs from the requirements model, and (b) to construct a new part of the model to cope with the changing of criteria for vertical separation between subsonic aircraft. Thus TR can be seen as a useful embedded component within a requirements validation regime for high integrity systems.

2 THE ATC DOMAIN

2.1 DOMAIN DESCRIPTION AND ACQUISITION

'Shanwick' is a large area of airspace in the eastern half of the North Atlantic, managed by air traffic control centres in Shannon, Ireland and Prestwick, Scotland. Controllers must organise this airspace daily, taking into account such factors as weather and the desired flight paths of aircraft companies. They plan the four dimensional flight profiles of aircraft crossing this airspace in good time before the aircraft reaches the boundary, and for this task require a precise definition of aircraft separation criteria, and an algorithm for predicting conflicts. The controllers are supported in their safety-critical work by a computer system which performs predication and resolution of conflicts between pairs of flight profiles, and our involvement came about as part of the research and development concerning the requirements specification of a replacement for their current flight data processing system.

In the FAROAS project, we created a precise requirements model (called the *CPS*) of the *conflict prediction* of aircraft flight profiles through the Shanwick airspace, together with a software support environment. Knowledge sources used were manuals of air traffic control, existing computer systems documentation, and air traffic control officers themselves. The current CPS contains a kernel of 300 - 400 axioms in msl representing aircraft profile separation criteria and a conflict prediction method; the total number of axioms in an instance of the model, which includes airspace and short term flight information for a day's set of profiles, exceeds two thousand. The model is structured into 23 sorts, and is enriched with real and natural numbers. An example of an axiom in the CPS is provided in Figure 1. This represents the condition for a vertical separation of 2,000 feet, where segments

```
(Segment1 and Segment2
are_subject_to_oceanic_cpr) =>
[(the_min_vertical_sep_Val_in_feet_required_for
Flight_level1 of Segment1
and Flight_level2 of Segment2) = 2000 <=>
[[ (both Segment1 and Segment2
are_flown_at_subsonic_speed)
& (one_or_both_of Flight_level1 and
Flight_level2 are_above FL 290) ] or
[(one_or_both_of Segment1 and Segment2
are_flown_at_supersonic_speed) &
(one_or_both_of Flight_level1 and
Flight_level2 are_at_or_below FL 430) ] ] ]
```

Figure 1: Condition for a Minimum Vertical Separation of 2000 feet

are roughly 'straight' components of an aircrafts profile. Either the two aircraft are both subsonic and are flying above FL 290 (29,000 feet) or one or both are supersonic and are flying at or below FL 430.

2.2 A CONVENTIONAL SUPPORT ENVIRONMENT

The CPS is highly structured, with axioms containing very complex conditions, but the support of an integrated tools environment alleviates its analysis and manipulation. In the FAROAS project diverse validation was carried out using tight syntactic checking, semantic internal consistency checks, expert inspection, simulation and batch testing. The most complex tool in the environment is a translator program which inputs the CPS (or more generally a set of wffs in msl), together with a syntactic definition of the tailored msl language expressed in grammar rules. It parses the wffs and outputs an animation of them by translating them into what we call 'EF' (execution form). This is similar to general clausal form, except clauses may contain nested negation and disjunction in their bodies. EF obeys the syntax rules of Prolog and is executable by a Prolog interpreter. This parsing and translation process takes less than 5 minutes for all of the CPS, and its translated form we term *CPS_{EF}*².

Flight profiles are input to the software environment as msl axioms and are translated into EF. Although in theory *any* part of the CPS can be tested, virtually all of the instances we obtained were for the 'top level'

²all software tools reported in this paper are implemented in Sicstus Prolog and were tested using a SUN SPARC station 4 processor with 32MB memory

conflict axiom defining the mixfix *conflict predicate*:

SegmentX of Profile1 and SegmentY of Profile2
are_in_oceanic_conflict

A day's worth (500 - 800) of cleared aircraft profiles, where each profile is cleared with (say) the last 20 cleared aircraft in chronological order, results in approximately 10,000 instances classified as false for the conflict axiom, where the SegmentX and SegmentY are existentially quantified variables representing segments of an aircraft's profile.

In the rest of this paper we use the following notation: a classified instance that is labelled true and which CPS_{EF} classifies as true is called truly positive ('TP'), and denoted e^{TP} , whereas one that executes to false is called falsely negative ('FN') and denoted e^{FN} . A classified instance labelled false which executes to false using CPS_{EF} is called truly negative ('TN'), whereas one that executes to true is called falsely positive ('FP'); these are denoted e^{TN} and e^{FP} respectively. Early phases of validation during the FAROAS project involving syntax checking and painstaking expert inspection increased the accuracy and completeness of the CPS so that dynamic testing of the conflict axiom resulted in a large number of TN, with a smaller (about 5 per cent) but significant number of FP. Although investigation of the set FP helped to find bugs, it became clear that more powerful tools for bug identification and removal were needed when building up and maintaining such a complex, precise domain model.

3 APPLICATION OF THEORY REVISION

3.1 RATIONALE

The principle objectives of the *current* project, IMPRESS, were to test the use of ML to help improve the quality (in terms of accuracy and completeness) of a formalised requirement specification written in msl and to increase the quality of the CPS itself. The focus was not only on bug removal but also on maintenance, to support the inevitable changes in the requirements model. Since we started with an existing symbolic domain model, the principle ML paradigm we decided to use was theory revision (Wrobel 1996). Our initial formulation was as follows:

Revisable theory: a subset of CPS_{EF} clauses. We can keep some parts of the CPS_{EF} immune or 'shielded' from the revision process, as they were ad-

equately validated using other processes. For example, it may be assumed that the 'top level' axioms, i.e. those defining the basics of separation in terms of vertical and horizontal dimensions, are correct. The target concept is the conflict predicate shown above.

Training Instances: The main source is a day's worth of cleared flight profiles supplied directly by the UK National Air Traffic Services. The conflict predicate can be executed, and when instantiated with pairs of cleared flight profiles should return false. The nature of the application skews the training somewhat as it is driven by FPs only. However, experiments have also been conducted with other, lower-level predicates as target concepts, such as those involved in vertical conflict. Instances associated with these conflicts are classified into FNs and TPs as well as TNs and FPs.

Learning Biases: the language used for the CPS_{EF} is strongly typed, which provides a useful constraint in the generalising or specialising of predicates. Also we assume a minimal revision bias: we know from other forms of validation that its structure mirrors the requirements domain, and so we assume only minimal revisions are necessary.

Given the general problem outlined above, we implemented a standard, simple TR algorithm with operators such as 'add antecedent' and 'delete clause'. However we only confirmed that a 'mainstream' approach to TR would be impracticable. Even given the biases, the potential space of revisions is enormous, and 'hill-climbing' with traditional TR operators appears out of the question. The CPS_{EF} executes the conflict axiom at an average rate of about one test per minute and results in a batch of tests taking perhaps days to execute!

We also investigated using TR tools, available via ftp, but came to the conclusion that we would need to build our own environment (West et al. 1996). This was based on the need for a flexible tool base given we were embarking on a research project, and the need for tool integration, particularly with our existing validation tools from the earlier FAROAS project. Moreover, the existing tools we examined were not powerful enough for our use. For example, FORTE (Richards and Mooney 1995), though well tested, could not cope with negation or functors. Both the latter are important features of the CPS. Also, while tools presented in the literature had been tested on theories of the order of 10's of predicates calls within a similar number of non-atomic clauses, the CPS_{EF} contains c.2,000 predicate calls within more than 300 non-atomic clauses.

3.2 ORDINAL SORTS

The key to our approach lay in the introduction of a further bias. Although the sorts comprising universes of objects are distinct, each sort can be characterised as either *ordered* or not (Birkhoff 1967). The sorts which are ordered are termed *ordinal* in this paper, and those which are not are termed *nominal*. Associated with each ordinal sort X is an *arbitrary* binary, transitive, ordering relationship we call ' \succeq '. Examples of ordinal sorts are *Flight Level*, *Time* and *Latitude*, where primitive order relations are for example 'is above', 'is later than', 'is west of'. Examples of nominal sorts are *Aircraft*, *Airspace*, *Segment*, *Profile*. Technical specifications such as the CPS include many references to ordinal sorts, and our experience in the validation phase had shown that very often clauses involving comparisons and limits were to blame. For example, of the 17 *primitive* order relations defined in the CPS's grammar, there are 204 occurrences of them within the current version of the CPS.

Each axiom in the CPS has as its *variable* domain:

$$X_1 \times \dots \times X_n \times D_1 \times \dots \times D_m, n, m \geq 0.$$

where each X_i is an ordinal sort and each D_j is a nominal sort. We will focus on axioms containing ' \succeq ': examples are $x \succeq a$, $x_1 \succeq x_2$, where x, x_1, x_2 are ordinal variables and a a constant, limiting value of some appropriate sort. The axiom involving ordering might be an equation defining a function, \mathcal{F} which returns different values for different subsets of its domain $X_1 \times \dots \times X_n \times D_1 \times \dots \times D_m$, or a predicate, \mathcal{P} . The statements involved in the definition of \mathcal{P} return 'true' or 'false' for different subsets of its domain $X_1 \times \dots \times X_n \times D_1 \times \dots \times D_m$. If we factor out the X_i from the D_j components, for each main predicate and function, for each tuple (d_1, \dots, d_m) of values, there is defined an n dimensional region $\mathcal{R}(d_1, \dots, d_m)$ – the domain of applicability of the predicate or function. For the remainder of the paper, we shall shorten this to \mathcal{R} .

When the CPS is translated to executable form, each axiom becomes a Prolog clause. The regions described above, for the main axioms, now become regions for Prolog clauses, where tuples of variables now become tuples of Prolog variables. In the case where a wff is an equation, its domain is extended by the returned term.

3.3 SIMPLE REVISIONS

Given a concept (for example the conflict predicate) and a set of positive instances of the concept, translated to EF, then the set of proof trees of the instances involve a set of clauses. Consider a clause C from this set, where its (Prolog) variables are the tuple $X_1, \dots, X_n, D_1, \dots, D_n$, where the X 's and D 's are ordinal and nominal respectively (where $n > 0$). Each instance of C is associated with an n -tuple of ordinal variables $(x_1, \dots, x_n) = \mathbf{x}$. We should expect positive instances to have $\mathbf{x} \in \mathcal{R}$. The region \mathcal{R} is defined by logical expressions $\mathcal{E}(\mathbf{x})$ involving ordinal variables \mathbf{x} and is not necessarily connected. In a similar manner a clause C which does *not* succeed and which is involved in a failed proof tree (or trace) of a negative instance will have $\mathbf{x} \notin \mathcal{R}$. In order for instances to *fail* where they previously succeeded, and vice-versa, then region \mathcal{R} is revised to become region \mathcal{R}' , for clause C . We classify revision operators that may change a clause containing an ordinal literal into two: *simple* and *composite*. Simple operators involve deletion and addition of antecedents from a clause, as in conventional TR, although the antecedents are restricted to occurrences of order relations. This kind of operator is mainly for finding and possibly correcting bugs in the model. For example, the condition $x \succeq y$ may be either removed or replaced by $y \succeq x$. This latter is akin (in 2-D geometrical terms) to examining reflections of the region about a straight line.

3.4 COMPOSITE REVISIONS

The second kind of revision is designed to clarify requirements involving complex conditions involving limiting values, which might not have been captured initially from the expert sources, and also to cope with changing requirements. We first deal with *specialisation*. Suppose C to be a candidate for revision, or *revision point*, where C contains antecedents of the form $x \succeq a$, a a constant. Further, suppose C succeeds with instances $\theta_i C$ in proofs of some training instances $e_i^{FP} \in FP$, with tuple \mathbf{x}_i the ordinal variables of $\theta_i C$. Suppose also that C is successful with instances $\phi_j C$ in proof tree of training instances e_j^{TP} ; the tuples \mathbf{y}_j are the ordinal variables of $\phi_j C$. Failure of C would ensure the removal of some instances e_i^{FP} and in order that C should fail, we need to *revise* \mathcal{R} to \mathcal{R}' .

$$\forall \mathbf{x}_i : \mathbf{x}_i \notin \mathcal{R}'.$$

However, in order that C should safely succeed for correctly classified instances, then tuples \mathbf{x} associated

with e^{TP} should *not* be removed from \mathcal{R} . Thus

$$\forall y_j : y_j \in \mathcal{R}'.$$

We calculate the two sets of tuples:

$$\begin{aligned} S^{FP} &= \{x_i \mid x_i \text{ ordinal variables of } \theta_i C\}, \\ S^{TP} &= \{y_j \mid y_j \text{ ordinal variables of } \phi_j C\}. \end{aligned} \quad (1)$$

This allows for the fact that the mis-classification of some/all of the instances e^{FP} may have arisen from another clause. Recalling that the variables of C are $\mathbf{x} = x_1 \dots x_n$, we denote the minimum and maximum values of variable component x_i of the S^{FP} variables by \min_i^{FP}, \max_i^{FP} respectively. In a similar manner the minimum and maximum values of components of the S^{TP} variables are respectively \min_i^{TP}, \max_i^{TP} .

We induce the following, that for instances $\theta_i C$ to fail, the new specialised region is \mathcal{R} less an n dimensional interval \mathcal{R}_{FP} bounded by \min_i^{FP}, \max_i^{FP} . We have

$$\begin{aligned} \mathcal{R}_{FP} &= \{(x_1 \dots x_n) \mid \min_1^{FP} \succeq x_1 \succeq \max_1^{FP} \wedge \\ &\dots \wedge \min_n^{FP} \succeq x_n \succeq \max_n^{FP}\} \end{aligned} \quad (2)$$

However for instances $\phi_j C$ to succeed, \mathcal{R}' must include an n dimensional interval \mathcal{R}_{TP} bounded by \min_i^{TP}, \max_i^{TP} :

$$\begin{aligned} \mathcal{R}_{TP} &= \{(x_1 \dots x_n) \mid \min_1^{TP} \succeq x_1 \succeq \max_1^{TP} \wedge \\ &\dots \wedge \min_n^{TP} \succeq x_n \succeq \max_n^{TP}\} \end{aligned} \quad (3)$$

We have

$$\mathcal{R}' = (\mathcal{R} \setminus \mathcal{R}_{FP}) \cup \mathcal{R}_{TP} \quad (4)$$

($\mathcal{R}_{TP} = \mathcal{R}_{FP}$ is the limiting case, where all of the mis-classified instances have arisen from another clause.) In order to accomplish the revision, we specialise the clause C as follows: every occurrence in the unrevised body of C of the logical expression $\mathcal{E}(x_1 \dots x_n)$, should be replaced in the revised body of C by $\mathcal{E}'(x_1 \dots x_n)$, which is defined:

$$\begin{aligned} (\mathcal{E}(x_1 \dots x_n) \wedge \neg (\min_1^{FP} \succeq x_1 \succeq \max_1^{FP} \wedge \\ \dots \wedge \min_n^{FP} \succeq x_n \succeq \max_n^{FP})) \\ \vee (\min_1^{TP} \succeq x_1 \succeq \max_1^{TP} \\ \dots \wedge \min_n^{TP} \succeq x_n \succeq \max_n^{TP}) \end{aligned} \quad (5)$$

Generalisation can be explained in a similar manner: in order for instances e^{FN} to succeed, their \mathbf{x} components are added to the region. However instances e^{TN}

must still fail. We calculate sets S^{FN}, S^{TN} and regions $\mathcal{R}_{FN}, \mathcal{R}_{TN}$ in an analogous manner to S^{FP}, S^{TP} in (1) and $\mathcal{R}_{FP}, \mathcal{R}_{TP}$ in (2).

We induce the following, that for some FN instances to succeed and *all* the TN instances to fail, then the new generalised region is

$$\mathcal{R}' = (\mathcal{R} \cup \mathcal{R}_{FN}) \setminus \mathcal{R}_{TN} \quad (6)$$

In order to accomplish this, we generalise the clause C , so that every occurrence in the unrevised body of C of the logical expression $\mathcal{E}(x_1 \dots x_n)$, should be replaced by $\mathcal{E}'(x_1 \dots x_n)$, in an analogous manner to (5). In the next section we explain how these simple and composite revisions were applied to CPS_{EF} .

4 EXPERIMENTS WITH TR TOOLS

We report experiments involving two kinds of data set:

1. The first data-set consists of training instances from a day's cleared flight profiles recorded in January 1995. This data was used with the object of testing our current techniques using 'simple' ordinal operators. When tested, the errors in the CPS as measured by this training set were 33 in 5070, having been previously reduced by other techniques. Use of TR with simple operators further reduced the errors to 1 in 5070.
2. 'Reduced separation for vertical minima' (RVSM) criteria have recently been introduced for certain types of aircraft in North Atlantic airspace. A days cleared flight profiles were provided (from April 1997), where clearance is subject to the new revised criteria (of flight levels) for vertical separations for pairs of aircraft. The new criteria involved flight level *intervals* for both aircraft and was not captured by our current theory. 'Simple' ordinal operators were not suitable for revisions of the type investigated, so this data-set was used for independently testing 'composite' ordinal operators. After the CPS was revised using simple operators, it was then re-revised using training instances from post-RVSM data and composite operators. All 121 errors resulting from the data cleared by the changed separation standard were eliminated by the method.

4.1 THE METHOD AND RESULTS

The method shown here is a general one for revision of a theory, Γ , containing significant ordinal variables,

although it is based on experiments with the CPS. We have implemented TR in a manner based on the 'geometrical' discussion above, and integrated it into our legacy software environment (McCluskey et al. 1995). This architecture has had to be both flexible and experimental in response to the inherent complexity of formal specifications of the size and expressiveness of the CPS. For example, we have had to develop a form of blame assignment that can cope with proof trees from general clausal form logic programmes. This involves first unfolding and then transforming negative literals using De Morgan's laws and is detailed in (West et al. 1997).

4.1.1 An Error Removal Experiment

The algorithm for simple ordinal revisions is based on conventional theory revision techniques and used hill-climbing based on the accuracy of the theory Γ . It is shown in Figure 2. The *potential* of a clause C is the number of instances in which it succeeds in a proof tree, and the negative potential is the number of instances in which it fails in a proof trace; this notion was used in the description of the FORTE tool (Richards and Mooney 1995).

1. Collect training set of instances of a concept, L , known to contain misclassified instances.
2. Classify training instances into TNs, FPs, FN's, TP's and calculate accuracy.
3. Run blame assignment on instances in FN giving set of potential-pairs, $P1$.
 $P1 = \{(C, N) \mid C \text{ revisable clause in } \Gamma \text{ and } N \text{ is the potential of } C\}$
 Find subset $OP1$ of $P1$:
 $OP1 = \{(C, N) \mid (C, N) \in P1 \wedge C \text{ contains an ordinal relation}\}.$
4. Repeat step 3 for instances in FP giving set of potential-pairs, $P2$, and subset $OP2$.
 Let $OP = OP1 \cup OP2$.
5. Revision points = $\{C \mid (C, N) \in\} OP$.
 Apply each simple TR operator to each revision point, in order of C with largest potential. Implement the best revision.
6. Repeat from step 2, unless a maximum accuracy has been reached.

Using a day's worth of training instances (cleared flight profiles) we obtained 33 FPs and 5037 TNs out of 5070 runs of the conflict axiom. Because of the complexity of the criteria the revision was accomplished by focusing the revision space to the longitudinal separation criteria (i.e. concept L in Figure 2) rather than from the initial training instances. L was selected by studying the output of blame assignment for all the FPs, and the generalised explanation output for individual FPs. Longitudinal separation values in minutes can be 5,6,7,8,9,10,15,20 or 30, and the CPS contains formalised criteria for all of these. 75 new training instances were generated from proof trees and proof traces in which a longitudinal separation value of 10 minutes was assigned to two aircraft at least one of which is flying at subsonic speed. The training instances included 25 FN and 50 TP, the concept being:

`the_basic_min_longitudinal_sep_Val_in_mins_`
`required_for(Segment1,Segment2) = 10.`

The TP's were generated by re-running the day's worth of instances, and identifying those in vertical conflict that gave a longitudinal separation of 10 minutes, but were *not* in overall conflict according to both air traffic control officers and the CPS_{EF} (thus lowering the possibility of noisy data). The FN's of concept L are derived directly from the 33 false positives from the conflict predicate. The algorithm using simple reverse and dropping conditions operators returned a new theory with two clauses altered by both the operators; after revision, 74 of the training instances were covered, and only 1 (FN) uncovered. Significantly, one of the clauses that was revised, defining the predicate:

`are_after_a_common_pt_from_which_profile_tracks`
`_are_same_or_diverging_thereafter_and_at_which`
`_both_aircraft_have_already_reported_by`

has been subsequently identified as an incorrect reading of an ATC Manual.

4.1.2 A Requirements Change Experiment

The method for implementing composite ordinal operators is shown in Figure 3. Steps 1 and 2 are similar to those of the simple operator. If after step 2, FN is larger than FP, then generalisation of a clause C occurs in steps 4b .. 8b in a similar manner. Note that the driver for the algorithm is the *stability* of the clauses in OP , rather than the increase in accuracy of Γ .

Figure 2: Algorithm for Simple Ordinal Operators

1. Collect training set of instances of a concept, known to contain misclassified instances. Initialise: D = Deleted clauses = $\{ \}$, A = added clauses = $\{ \}$.
2. Classify training instances into TNs, FPs, FN's, TPs and calculate accuracy.
- 3a. Specialise Γ in a manner indicated by steps 3a .. 8a. Run blame assignment on instances in FP giving set of potential-pairs, P .
 $P = \{(C, N) \mid C \text{ revisable clause in } \Gamma \text{ and } N \text{ is the potential of } C\}$.
 Find subset OP of P :
 $OP = \{(C, N) \mid (C, N) \in P \wedge C \text{ contains an ordinal relation}\}$.
- 4a. Select pair (C, N) where N is maximum of $\{N \mid (C, N) \in\} OP$.
- 5a. Calculate the n dimensional regions $\mathcal{R}_{FN}, \mathcal{R}_{TN}$ defined by (1), (2) and (3).
- 6a. If $\mathcal{R}_{FN}, \mathcal{R}_{TN}$ are not equal
 set head of $C' :=$ head of C ;
 set body of $C' :=$ body of C with \mathcal{E} replaced by \mathcal{E}' from (5)
 else
 delete C from OP and repeat from step 4a.
- 7a. Replace C with C' and calculate accuracy.
- 8a. $D' = D \cup C$; $A' = A \cup C'$.
9. Repeat from step 1 until OP is stable or accuracy is 100 %.

Figure 3: Algorithm for Composite Ordinal Operators.

Because of the safety-critical nature of the application, and the fact that some data values may occur only rarely, it is necessary to check that a clause $C \in A$ originally arising from a function, remains defined over its intended domain. If this is the case, a *post-processing* phase is necessary.

204 training instances (classified according to post-RVSM criteria) of the conflict axiom were used to revise the CPS using the algorithm in Figure 3. However, revisions were confined to ordinals of the form 'is_above'. When tested, there were found to be 121 FP instances, and 83 FN instances. The 'blame assignment pinpointed the clause 'the_min_vertical_sep_Val_in_feet_required_for(A, B, C, D, 2000)'

as a revision point and the results are shown below. (The 'limitvar' predicate is a device for marking variable occurrences.) As can be seen, for supersonic aircraft, the criteria is unaltered. The criteria for a vertical separation of 2000 feet are specialised; they exclude the region where both flight levels are between FL 330 and FL 370 as shown in the following result:

lengths of FN, FP, TN, TP

0 121 83 0

%% set P.

[potential(1,121),potential(2,121), ...,
potential(23,1),potential(26,121), ..]

%% list of revision points

[26]

New_accuracy = 100.0, Old_accuracy = 40.686

%%revised code for 2000

the_min_vertical_sep_Val_in_feet_required_for(
A, B, C, D, 2000) :-

(both_are_flown_at_subsonic_speed(B, D),

(A is_above fl(290), limitvar(1),
((not__(A is_at_or_above fl(330))
; not__(A is_at_or_below fl(370)))
; not__(C is_at_or_above fl(330))
; not__(C is_at_or_below fl(370)))

; C is_above fl(290), limitvar(2),
((not__(A is_at_or_above fl(330))
; not__(A is_at_or_below fl(370))
)

; not__(C is_at_or_above fl(330))
; not__(C is_at_or_below fl(370)))

;

one_or_both_of_are_flown_at_supersonic_speed(
B, D),

(A is_at_or_below fl(430), limitvar(3),

; C is_at_or_below fl(430), limitvar(4))),!.

5 RELATED WORK

Some recent work has pointed to the similarities between the validation of requirements models and knowledge based systems development (McCluskey et al. 1996; Shaw and Gaines 1996), and hence the area of Knowledge Base Refinement (KBR) is related to our work. A detailed comparison of validation in software engineering and KBS is given in reference (Vermesan and Bench-Capon 1995), and the state of the art in *automated* KBS validation is surveyed in refer-

ence (Zlatareva and Preece 1994).

As far as we are aware our work is the first to apply machine learning techniques to formal specifications of requirements, although, as mentioned above, work most related to our own occurs in the field of KBR. Both areas have to adopt strategies to overcome the complexity pitfalls surrounding the use of TR (where theoretical results suggest that no polynomial algorithm exists to perform global optimisation in hill climbing algorithms (Greiner 1995)). In KRUST (Palmer and Craw 1996), for example, test cases are used one at a time to refine the KBS, in contrast to our focusing procedure, which uses multiple examples and a form of statistical blame assignment. In MOBAL, an environment for knowledge acquisition that has been used with a large security rule base, TR is also used but in restrained fashion and with limited success (see (Sommer et al. 1994) page 453). Experience with MOBAL is consistent with our experience that ML tools work well in the context of a *diverse* tools environment.

Imperfect theory refinement techniques have been well researched in the machine learning literature, including reviews (Wrobel 1996), and a text relating ML to Software Engineering (Bergadano and Gunetti 1996). The case where theories represent planning domains is described in reference (Tae and Cook 1996) and the case where theories are posed as Horn Clause models is described in reference (Richards and Mooney 1995). Machine learning in domains containing significant numerical components has previously been accomplished by using neural networks (Opitz and Shavlik 1997). *Constraint Inductive Logic Programming* (Anthony and Frisch 1997; Sebag and Rouveirol 1996) has been utilised for generalisation and specialisation of numerical predicates. Theory Patching (Argamon-Engelson and Koppel 1998) is described as a type of TR in which revisions are made to individual components of the theory. (The concern of the latter paper is to determine which classes of logical domain theories the theory patching problem is tractable.) Theory patching compares with our work on focusing on *ordinal revisions* and on shielding clauses which are not to be revised.

6 CONCLUSIONS AND FURTHER WORK

In this paper we have reported the application of theory revision techniques to the validation and maintenance of a substantial 'theory', the formal requirements model of an air traffic control application. The

model is encoded in msl, is customised by a generative grammar, animated by a Prolog generator, and can be analysed using an integrated environment supporting a diverse range of validation techniques (McCluskey 1997). After overcoming problems to do with blame assignment in general clause form programs (West et al. 1997), we developed the method whereby batches of tests were used by blame assignment, and single tests were used by explanation-based tools, to identify axioms sets in which bugs were likely to reside. After acquiring classified instances for these faulty components, we used theory revision operators, targeting comparison operators acting on ordinal sorts, to identify and remove the bugs. Here we have shown two different experiments where bugs were identified and removed, and a new part of the model was induced. The project started with an error rate for the conflict predicate of several hundred errors per 10,000 tests. The application of ML techniques in general has lead us to establish the cause of all the errors shown up in our initial tests, and the error rates using code generated from the current version of our model have been cut by 2 orders of magnitude. Having said this, our success in fielding TR seems to depend on correctly predicting how fundamental the revisions are, and having the machinery available to bring about such a level of revision.

Many problems for future work remain, however. Most outstanding is the generalisation of our environment so that other customised msl models can be created and analysed using ML tools. Secondly, the TR algorithms for simple and composite revisions need to be further refined and perhaps merged. Also, the implications of using blame assignment which takes into account negative literals in proof trees needs to be fully evaluated.

Acknowledgements

The IMPRESS project is supported by an EPSRC grant, number GR/K73152. We would like to acknowledge the help of Chris Bryant, who implemented some of the theory revision tools, Julie Porteous, for help in the initial stages of IMPRESS, and Julia Sonander of NATS, for supplying aircraft profile data. Further, we would like to thank the referees for helpful suggestions, and additional references, which have improved this paper.

References

- Anthony, S. and A. Frisch (1997). Generating Numerical Literals during Refinement. In N. Lavrac and S. Dzeroski (Eds.), *Inductive Logic Programming: Proceedings of the 7th International Workshop, ILP-97*, Volume 1297 of *Lecture Notes in Artificial Intelli-*

- gence, pp. 61 – 76. Springer-Verlag.
- Argamon-Engelson, S. and M. Koppel (1998). Tractability of theory patching. *Journal of Artificial Intelligence Research* 8, 39–65.
- Bergadano, F. and D. Gunetti (1996). *Inductive Logic Programming, From Machine Learning to Software Engineering*. Cambridge, Massachusetts, US: MIT Press.
- Birkhoff, G. (1967). *Lattice Theory*. (Third ed.). American Mathematical Society.
- Greiner, R. (1995). The complexity of theory revision. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.
- McCluskey, T. L. (1997). An integrated environment for the conflict prediction specification. Technical Report impress/2/03/1, School of Computing and Mathematics, University of Huddersfield, UK.
- McCluskey, T. L., J. M. Porteous, Y. Naik, C. Taylor, and S. Jones (1995). A requirements capture method and its use in an air traffic control application. *Software - Practice and Experience* 25(1), 47–71.
- McCluskey, T. L., J. M. Porteous, M. M. West, and C. H. Bryant (1996, September). The validation of formal specifications of requirements. In *Proceedings of the BCS-FACS Northern Formal Methods Workshop*, Ilkley, UK. Electronic Workshops in Computing Series, Springer.
- Meinke, K. and J. Tucker (1993). *Many Sorted Logic and Its Applications*. Wiley.
- Opitz, D. W. and J. W. Shavlik (1997). Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research* 6, 177–209.
- Palmer, G. J. and S. Craw (1996). The role of test cases in automated knowledge refinement. In *ES96: The Sixteenth Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge, England, pp. 75–90.
- Richards, B. L. and R. J. Mooney (1995, May). Automated refinement of first-order horn-clause domain theories. *Machine Learning* 19(2), 95–131.
- Sebag, M. and C. Rouveirol (1996). Constraint inductive logic programming. In L. De Raedt (Ed.), *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pp. 277 – 294. IOS Press.
- Shaw, L. and B. Gaines (1996). Requirements acquisition. *Software Engineering Journal* 11, 149–165.
- Sommer, E., K. Morik, J. M. Andre, and M. Uszynski (1994). What online machine learning can do for knowledge acquisition - a case-study. *Knowledge Acquisition* 6(4), 435–460.
- Tae, K. and D. Cook (1996). Experimental knowledge acquisition for planning. In *Proceedings of the 13th International Conference on Machine Learning: ML'96*, pp. 480–488.
- Vermesan, A. and T. Bench-Capon (1995). Techniques for the verification and validation of knowledge-based systems: a survey based on the symbol/knowledge level distinction. *Software Testing, Verification and Reliability* 5, 233–271.
- West, M. M., C. H. Bryant, and T. L. McCluskey (1997). Transforming general program proofs: A meta interpreter which expands negative literals. In *Proceedings: LOPSTR '97*, Leuven, Belgium.
- West, M. M., C. H. Bryant, T. L. McCluskey, and J. M. Porteous (1996). The use of machine learning in the validation of a formal requirements specification: the work of IMPRESS. Technical Report impress/1/01/1, School of Computing and Mathematics, University of Huddersfield, UK.
- Wrobel, S. (1996). First order theory revision. In L. De Raedt (Ed.), *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pp. 14–33. IOS Press.
- Zlatareva, N. and A. Precce (1994). State-of-the-art in automated validation of knowledge-based systems. *Expert Systems with Applications* 7(2), 151–167.

Stochastic Resonance with Adaptive Fuzzy Systems

Sanya Mitaim and Bart Kosko
 Signal and Image Processing Institute
 Department of Electrical Engineering—Systems
 University of Southern California
 Los Angeles, California 90089-2564

Abstract

Adaptive systems can learn to add an optimal amount of noise to some nonlinear feedback systems. Noise can improve the signal-to-noise ratio of many nonlinear dynamical systems. This "stochastic resonance" effect occurs in a wide range of physical and biological systems. The SR effect may also occur in engineering systems in signal processing, communications, and control. The noise energy can enhance the faint periodic signals or faint broadband signals that force the dynamical systems. Most SR studies assume full knowledge of a system's dynamics and its noise and signal structure. Fuzzy and other adaptive systems can learn to induce SR based only on samples from the process. These samples can tune a fuzzy system's if-then rules so that the fuzzy system approximates the dynamical system and its noise response. The paper derives the SR optimality conditions that any stochastic learning system should try to achieve. The adaptive system learns the SR effect as the system performs a stochastic gradient ascent on the signal-to-noise ratio. The stochastic learning scheme does not depend on a fuzzy system or any other adaptive system. The learning process is slow and noisy and can require heavy computation. Robust noise suppressors can improve the learning process when we can estimate the impulsiveness of the noise or of other learning terms. Simulations test this SR learning scheme on the popular quartic-bistable dynamical system and on other dynamical systems for many types of noise. Simulations suggest that fuzzy techniques and perhaps other "intelligent" techniques can induce SR in many cases when users cannot state the exact form of the dynamical systems.

1 STOCHASTIC RESONANCE

Noise can sometimes enhance a signal as well as corrupt it. This fact may seem at odds with almost a century of effort in signal processing to filter noise or to mask or cancel it. But noise is itself a signal and a free source of energy. Noise can amplify a faint signal in some feedback nonlinear systems even though too much noise can swamp the signal. This implies that a system's optimal noise level need not be zero noise. It also suggests that nonlinear signal systems with nonzero-noise optima may be the rule rather than the exception.

Stochastic resonance (SR) [2, 3, 16] occurs when noise enhances an external forcing signal in a nonlinear dynamical system. SR occurs in a signal system if and only if the system has a nonzero noise optimum. The classic SR signature is a signal-to-noise ratio (SNR) that is not monotone. Figure 1 shows the SR effect for the popular quartic bistable dynamical system [2, 3]. The SNR rises to a maximum and then falls as the variance of the additive white noise grows. More complex systems may have multimodal SNRs.

SR holds promise for the design of engineering systems in a wide range of applications. Engineers may want to shape the noise background of a fixed signal pattern to exploit the SR effect. Or they may want to adapt their signals to exploit a fixed noise background. Engineers now add noise to some systems to improve how humans perceive signals [12, 14]. Some control schemes add a noise-like dither to improve system performance [18].

The study of SR has emerged largely from physics and biology. The awkward term "stochastic resonance" stems from a 1981 article in which physicists observed "the cooperative effect between internal mechanism and the external periodic forcing" in some nonlinear dynamical systems [2]. Scientists soon explored SR in climate models [17] to explain how noise could induce periodic ice ages [1]. They conjectured that global or other noise sources could amplify small periodic vari-

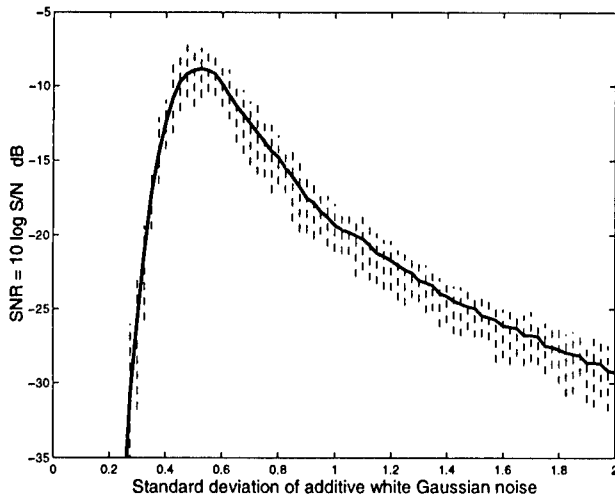


Figure 1: The non-monotonic signature of stochastic resonance. The graph shows the smoothed output signal-to-noise ratio of the noisy signal-forced quartic bistable system $\dot{x} = f(x) + s(t) + n(t) = x - x^3 + s(t) + n(t)$. The vertical dashed lines show the absolute deviation between the smallest and largest outliers in each sample average of 20 outcomes. The system has a nonzero noise optimum and thus shows the SR effect. The Gaussian noise $n(t)$ adds to the external forcing narrowband signal $s(t) = \varepsilon \sin \omega_0 t$. Other systems can use multiplicative noise or use non-Gaussian noise [4].

ations in the Earth's orbit. This might explain the observed 100,000 year primary cycle of the Earth's ice ages. Physicists have since found stronger evidence of SR in various systems [11, 16, 19].

Below we explore how to learn the SR effect with adaptive systems in general and with adaptive fuzzy function approximators [9] in particular. Neural-like learning laws tune and move the fuzzy rule patches as they tune the shape of the fuzzy sets that make up the rule patches. The learning laws use input-output data from the sampled noisy dynamical system. The rule patches move quickly to cover optimal or near-optimal regions of the function (such as its extrema). Fuzzy systems achieve their patch-covering approximation at the high cost of rule explosion [9]. The number of rules grows exponentially with the state-space dimension of the fuzzy system. We stress that our SR learning laws can also tune non-fuzzy adaptive systems. Our first goal was to show that adaptive systems can learn to shape the input noise and perhaps shape other terms to achieve SR in the main closed-form dynamical systems that scientists have shown produce the SR effect. Our second goal was to suggest through these simulation experiments that adaptive fuzzy systems or other model-free approximators might achieve SR in the more complex dynamical systems that defy easy

math modeling or measurement.

This paper presents three main results. The first and central result is that a system can learn the SR effect if it performs a stochastic gradient ascent on the signal-to-noise ratio $\text{SNR} = S/N$. Then the random noise gradient $\frac{\partial \text{SNR}}{\partial \sigma}$ can tune the parameters in any adaptive system through a slow type of stochastic approximation. The second result is that the SNR first-order condition for an extremum has the ratio form $\frac{S}{N} = \frac{S'}{N'}$ for $S' = \frac{\partial S}{\partial \sigma}$. The term $\frac{S'}{N'}$ can produce impulsive or even Cauchy noise that can destabilize the stochastic gradient ascent. Time lags in the training process can compound this impulsiveness. The third result is that a Cauchy-based noise suppressor from the theory of robust statistics can often reduce the impulsiveness of the noise gradient $\frac{\partial \text{SNR}}{\partial \sigma}$ and thus improve the learning process.

2 ADDITIVE FUZZY SYSTEMS & FUNCTION APPROXIMATION

A fuzzy system $F : R^n \rightarrow R^p$ stores m rules of the word form "If $X = A_j$ Then $Y = B_j$ " or the patch form $A_j \times B_j \subset X \times Y = R^n \times R^p$. The if-part fuzzy sets $A_j \subset R^n$ and then-part fuzzy sets $B_j \subset R^p$ have set functions $a_j : R^n \rightarrow [0, 1]$ and $b_j : R^p \rightarrow [0, 1]$. Generalized fuzzy sets map to intervals other than $[0, 1]$. The scalar sinc set functions in Figure 6 map real inputs to "membership degrees" in the bipolar range $[-0.217, 1]$. The system design must take care when these negative set values enter the SAM ratio in (2). The system can use the joint set function a_j or some factored form such as $a_j(x) = a_j^1(x_1) \cdots a_j^n(x_n)$ or $a_j(x) = \min(a_j^1(x_1), \dots, a_j^n(x_n))$ or any other conjunctive form for input vector $x = (x_1, \dots, x_n) \in R^n$ [9]. An additive fuzzy system [9] sums the "fired" then-part sets B_j' :

$$B(x) = \sum_{j=1}^m w_j B_j' = \sum_{j=1}^m w_j a_j(x) B_j. \quad (1)$$

Figure 2a shows the parallel fire-and-sum structure of the standard additive model (SAM). These nonlinear systems can uniformly approximate any continuous (or bounded measurable) function f on a compact domain [9]. Engineers often apply fuzzy systems to problems of control but fuzzy systems can also apply to problems of communication and signal processing [9] and other fields.

Figure 2b shows how three rule patches can cover part of the graph of a scalar function $f : R \rightarrow R$. The patch-cover structure implies that fuzzy systems $F : R^n \rightarrow R^p$ suffer from *rule explosion* in high dimensions. A fuzzy system F needs on the order of k^{n+p-1} rules to cover the graph and thus to approximate a

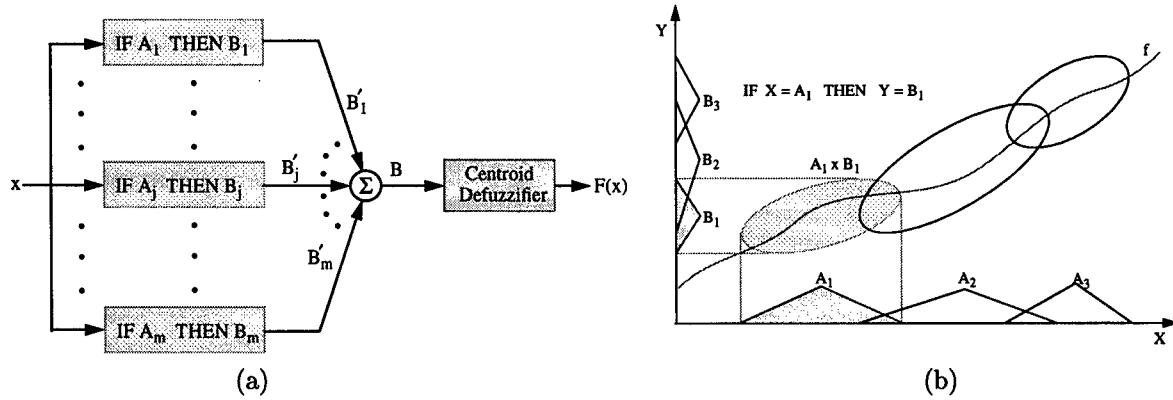


Figure 2: Feedforward fuzzy function approximator. (a) The parallel associative structure of the additive fuzzy system $F : R^n \rightarrow R^p$ with m rules. Each input $x_0 \in R^n$ enters the system F as a numerical vector. At the set level x_0 acts as a delta pulse $\delta(x - x_0)$ that combs the if-part fuzzy sets A_j and gives the m set values $a_j(x_0) = \int_{R^n} \delta(x - x_0) a_j(x) dx$. The set values “fire” or scale the then-part fuzzy sets B_j to give B'_j . A standard additive model (SAM) scales each B_j with $a_j(x)$. Then the system sums the B'_j sets to give the output “set” B . The system output $F(x_0)$ is the centroid of B . (b) Fuzzy rules define Cartesian rule patches $A_j \times B_j$ in the input-output space and cover the graph of the approximand f .

vector function $f : R^n \rightarrow R^p$. Optimal rules can help deal with the exponential rule explosion. Lone or local mean-squared optimal rule patches cover the extrema of the approximand f [9]. They “patch the bumps.” Better learning schemes move rule patches to or near extrema and then fill in between extrema with extra rule patches if the rule budget allows.

The scaling choice $B'_j = a_j(x)B_j$ gives a *standard additive model* or SAM. Taking the centroid of $B(x)$ in (1) gives the following SAM ratio [9]

$$F(x) = \frac{\sum_{j=1}^m w_j a_j(x) V_j c_j}{\sum_{j=1}^m w_j a_j(x) V_j} = \sum_{j=1}^m p_j(x) c_j. \quad (2)$$

The if-part fuzzy sets $A_j \subset R^n$ has set functions $a_j : R^n \rightarrow [0, 1]$. The then-part sets $B_j \subset R^p$ has finite positive volume or area V_j and centroid or its center of mass c_j . The convex weights $p_1(x), \dots, p_m(x)$ have the form $p_j(x) = \frac{w_j a_j(x) V_j}{\sum_{i=1}^m w_i a_i(x) V_i}$. The convex coefficients $p_j(x)$ change with each input vector x . We can ignore the rule weights w_j if we put $w_1 = \dots = w_m > 0$.

3 SR LEARNING AND EQUILIBRIUM

The scalar standard additive model (SAM) [9] fuzzy system $F : R^n \rightarrow R$ can learn the SR pattern of optimum noise of an unknown dynamical system if it uses enough rules and if it samples enough data from a dynamical system that stochastically resonates. Below we derive a gradient-based learning law that tunes the SAM parameters to achieve SR from samples of

system dynamics. It can also tune the parameters in other adaptive systems. We first define a practical SNR measure in terms of discrete Fourier transforms. Other SR measures can give other learning laws.

3.1 THE SNR IN NONLINEAR SYSTEMS

Suppose a nonlinear dynamical system has a sinewave forcing function $s(t)$ of known frequency f_0 Hz. We search the sinusoidal part $r(t)$ of the output $y(t)$ for the known frequency f_0 but unknown amplitude and phase in the system output response $y(t)$. The “noisy signal” $y(t)$ has the form of “signal” plus “noise”:

$$y_t = r_t + n_t. \quad (3)$$

The signal-to-noise ratio (SNR) at the output is the spectral ratio of the energy of $\{r_t\}$ to the energy of $\{n_t\}$. We assume that the signal $s(t)$ is always present. This ignores the important problem of signal detection but lets us focus on learning the SR effect.

We define the SNR measure as

$$\text{SNR} = \frac{S}{N} = \frac{S}{P - S}. \quad (4)$$

Here $S = 2|Y[k_0]|^2$, $P = \sum_{k=0}^{L-1} |Y[k]|^2$, and $Y[k]$ is the L -point discrete Fourier transform (DFT) of y_n :

$$Y[k] = \sum_{t=0}^{L-1} y_t e^{-i \frac{2\pi k}{L} t}. \quad (5)$$

We assume that the discrete frequency $k_0 = f_0 L T_s > 0$ is an integer for sampling rate $1/T_s$ and $\omega_0 = 2\pi f_0$. We also assume that there is no aliasing due to sampling.

Then we can show that for large L the SNR measure in (4) tends to the standard definition of SNR as a ratio of variances:

$$\text{Theorem: } \text{SNR} = \frac{2|Y[k_0]|^2}{\sum_{k=0}^{L-1} |Y[k]|^2 - 2|Y[k_0]|^2}$$

$$\rightarrow \frac{\sigma_r^2}{\sigma_n^2} = \frac{A^2/2}{\sigma_n^2} = \frac{\frac{1}{L} \sum_{t=0}^{L-1} r_t^2}{\frac{1}{L} \sum_{t=0}^{L-1} n_t^2}.$$

Here $\sigma_n^2 = \text{Var}(n) = E[n^2] < \infty$ and $\sigma_r^2 = \frac{1}{T} \int_0^T (A \sin \omega_0 t)^2 dt = A^2/2$.

3.2 SR LEARNING AND OPTIMALITY

An adaptive system can learn a SR noise pattern that maximizes a dynamical system's SNR. The learning law updates a parameter m_j of a SAM fuzzy system (or of any other adaptive system) at time step n with the deterministic law

$$m_j(n+1) = m_j(n) + \mu_n \frac{\partial E[\text{SNR}]}{\partial m_j} \quad (6)$$

for learning coefficients $\{\mu_n\}$. This is gradient *ascent* learning. We assume that the first-order moment of the SNR exists and is finite. We seldom know the probability structure or the expectation of the SNR. So we estimate this expectation with its random realization at each time step: $E[\text{SNR}] \approx \text{SNR}$. This gives the *stochastic* gradient learning law

$$m_j(n+1) = m_j(n) + \mu_n \frac{\partial \text{SNR}}{\partial m_j} \quad (7)$$

or simple random hill climbing. We assume the chain rule holds (at least approximately) to give

$$\frac{\partial \text{SNR}}{\partial m_j} = \frac{\partial \text{SNR}}{\partial \sigma} \frac{\partial \sigma}{\partial m_j} = \frac{\partial \text{SNR}}{\partial \sigma} \frac{\partial F}{\partial m_j}. \quad (8)$$

Here σ is the noise level or standard deviation of the forcing noise term $n(t)$. We want the SAM or other adaptive system F to approximate the optimal noise level σ_{opt} for any input signal or initial condition of the dynamical system: $F \approx \sigma_{opt}$. We then use σ and F interchangeably in (8). The term $\frac{\partial F}{\partial m_j}$ shows how any adaptive system F depends on its j th parameter m_j . We again assume that the chain rule holds to get

$$\frac{\partial \text{SNR}}{\partial \sigma} = \frac{\partial \text{SNR}}{\partial S} \frac{\partial S}{\partial \sigma} + \frac{\partial \text{SNR}}{\partial N} \frac{\partial N}{\partial \sigma}. \quad (9)$$

Then $\text{SNR} = 10 \log S/N$ implies that

$$\frac{\partial \text{SNR}}{\partial S} = \frac{\partial}{\partial S} 10 \log \frac{S}{N} = (10 \log e) \frac{1}{S} \quad (10)$$

$$\frac{\partial \text{SNR}}{\partial N} = \frac{\partial}{\partial N} 10 \log \frac{S}{N} = -(10 \log e) \frac{1}{N}. \quad (11)$$

for base-10 logarithm. We next put (10)-(11) into (9) to get the log term that drives SR learning:

$$\frac{\partial \text{SNR}}{\partial \sigma} = (10 \log e) \left(\frac{1}{S} \frac{\partial S}{\partial \sigma} - \frac{1}{N} \frac{\partial N}{\partial \sigma} \right). \quad (12)$$

The right side of (12) leads to the first-order condition for an SNR extremum:

$$\frac{1}{S} \frac{\partial S}{\partial \sigma} - \frac{1}{N} \frac{\partial N}{\partial \sigma} = 0 \quad \text{or} \quad \frac{S}{N} = \frac{S'}{N'} \quad (13)$$

when the partial derivatives of S and N with respect to σ are not zero at $\sigma = \sigma_{opt}$. Equation (13) gives a necessary condition for the SR maximum. The result (13) says that at SR the ratio of the rate of changes of S and N must equal the ratio of S and N . But (13) holds only in a stochastic sense for sufficiently well-behaved random processes. The second-order condition for an SR maximum is

$$0 > \frac{\partial^2 \text{SNR}}{\partial \sigma^2} = \frac{\partial}{\partial \sigma} (10 \log e) \left[\frac{1}{S} \frac{\partial S}{\partial \sigma} - \frac{1}{N} \frac{\partial N}{\partial \sigma} \right] \quad (14)$$

$$= (10 \log e) \left[\frac{1}{S} \frac{\partial^2 S}{\partial \sigma^2} - \frac{1}{S^2} \left(\frac{\partial S}{\partial \sigma} \right)^2 - \frac{1}{N} \frac{\partial^2 N}{\partial \sigma^2} + \frac{1}{N^2} \left(\frac{\partial N}{\partial \sigma} \right)^2 \right] \quad (15)$$

$$= (10 \log e) \left[\frac{1}{S} \frac{\partial^2 S}{\partial \sigma^2} - \frac{1}{N} \frac{\partial^2 N}{\partial \sigma^2} \right] \quad (16)$$

or $\frac{S''}{S} < \frac{N''}{N}$. The last equality follows from the first-order condition $\frac{1}{S} \frac{\partial S}{\partial \sigma} - \frac{1}{N} \frac{\partial N}{\partial \sigma} = 0$ or $\frac{S'}{S} = \frac{N'}{N}$ since then $\frac{(S')^2}{S^2} = \frac{(N')^2}{N^2}$. A like result holds for $\text{SNR} = S/N$. These first- and second-order conditions show how the signal power S and noise power N relate to each other and to their derivatives at the SR maximum.

We now derive the SR learning laws in terms of DFTs. We can approximate $\frac{\partial S_n}{\partial \sigma_n}$ and $\frac{\partial N_n}{\partial \sigma_n}$ with a ratio of time differences at each iteration n :

$$\frac{\partial S_n}{\partial \sigma_n} \approx \frac{\Delta S_n}{\Delta \sigma_n} = \frac{S_n - S_{n-1}}{\sigma_n - \sigma_{n-1}} \quad (17)$$

$$\frac{\partial N_n}{\partial \sigma_n} \approx \frac{\Delta N_n}{\Delta \sigma_n} = \frac{N_n - N_{n-1}}{\sigma_n - \sigma_{n-1}}. \quad (18)$$

Then put (17) and (18) into (9) to get the stochastic gradient learning law:

$$m_j^{n+1} = m_j^n + \mu_n \frac{\partial \text{SNR}_n}{\partial m_j} \quad (19)$$

$$= m_j^n + \mu_n \frac{\partial \text{SNR}_n}{\partial \sigma_n} \frac{\partial F}{\partial m_j} \quad (20)$$

$$= m_j^n + \mu_n \left(\frac{1}{S_n} \frac{\partial S_n}{\partial \sigma_n} - \frac{1}{N_n} \frac{\partial N_n}{\partial \sigma_n} \right) \frac{\partial F}{\partial m_j}. \quad (21)$$

Below we derive the last partial derivative $\frac{\partial F}{\partial m_j}$ in the chain-rule expansion (8) for all SAM fuzzy parameters.

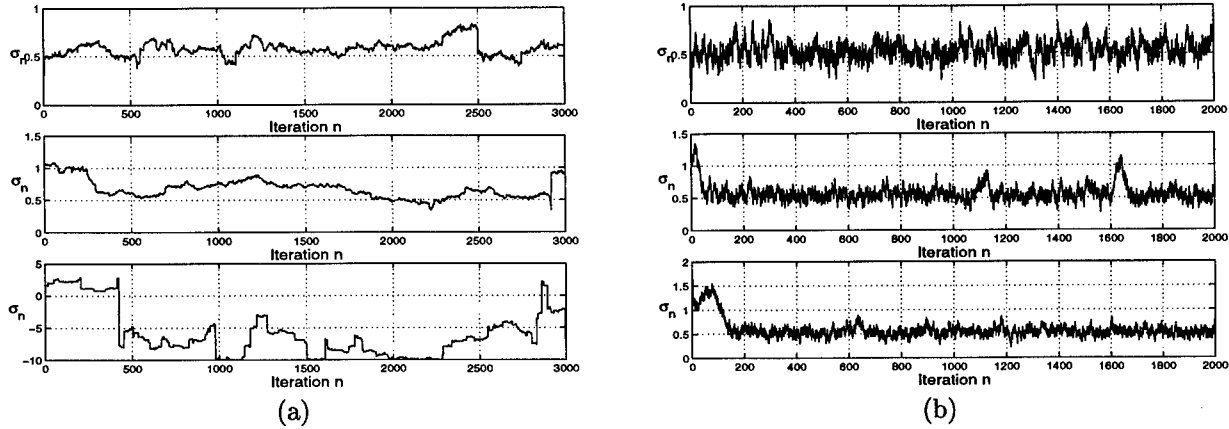


Figure 3: Learning paths of σ_n for the quartic bistable system (30)-(31). The input sinusoid signal function is $s(t) = 0.1 \sin 2\pi(0.01)t$. The optimum noise intensity lies near $\sigma = 0.5$ from the SNR-noise profile in Figure 1. (a) Impulsive effects on learning paths of noise level σ_n with different initial values. The paths of σ_n do not converge to the optimum noise. This stems from the impulsiveness of the derivative term $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$ in the SR learning law. (b) Learning paths of σ_n with the Cauchy noise suppressor ϕ . The term $\phi(\frac{\partial \text{SNR}_n}{\partial \sigma_n})$ replaces $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$ in the SR learning law as in (36). The paths of σ_n wander in a Brownian-like motion around the optimum noise. The suppressor function ϕ makes the learning algorithm more robust against impulsive shocks.

This is again the step where users can insert other adaptive function approximators F and derive learning laws for their parameters m_j by expanding $\frac{\partial F}{\partial m_j}$. The chain rule gives the partial derivatives

$$\frac{\partial F}{\partial c_j} = \frac{w_j a_j(x) V_j}{\sum_{i=1}^m w_i a_i(x) V_i} = p_j(x) \quad (22)$$

$$\frac{\partial F}{\partial V_j} = \frac{w_j a_j(x) [c_j - F(x)]}{\sum_{i=1}^m w_i a_i(x) V_i} = \frac{p_j(x)}{V_j} [c_j - F(x)] \quad (23)$$

$$\frac{\partial F}{\partial m_j} = \frac{\partial F}{\partial a_j} \frac{\partial a_j}{\partial m_j} \quad \text{and} \quad \frac{\partial F}{\partial d_j} = \frac{\partial F}{\partial a_j} \frac{\partial a_j}{\partial d_j} \quad (24)$$

where

$$\frac{\partial F}{\partial a_j} = [c_j - F(x)] \frac{p_j(x)}{a_j(x)}. \quad (25)$$

We used the sinc set functions [9, 13] in our simulation. The sinc set function has the form $a_j(x) = \sin\left(\frac{x-m_j}{d_j}\right) / \left(\frac{x-m_j}{d_j}\right)$. So the partial derivatives are

$$\frac{\partial a_j}{\partial m_j} = \begin{cases} \left(a_j(x) - \cos\left(\frac{x-m_j}{d_j}\right)\right) \frac{1}{x-m_j} & \text{for } x \neq m_j \\ 0 & \text{for } x = m_j \end{cases} \quad (26)$$

$$\frac{\partial a_j}{\partial d_j} = \left(a_j(x) - \cos\left(\frac{x-m_j}{d_j}\right)\right) \frac{1}{d_j}. \quad (27)$$

We used small but constant learning rates in most simulations.

4 SIMULATION RESULTS

This section shows how the stochastic SR learning laws in Section 3 tend to find the optimal noise levels. The

learning process updates the noise parameter σ_n at each sample time n . The learning process is noisy and may not be stable due to the impulsiveness of the random gradient $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$. We used a Cauchy noise suppressor from the theory of robust statistics [8] to stabilize the learning process. Then sample paths of σ_n converged and wander about the optimal values if the initial values were close to the optimum.

The response of a system depends on its dynamics and on the nature of its input signals. We applied the SNR measure to the quartic bistable system with sinusoidal inputs. Future research may extend SR learning to wideband input signals. Figure 7a shows how the optimum noise level varies for each input sinewave in the quartic bistable system. The learning process samples the system's input-output response as it learns the optimum noise. It does not make direct use of the equation that underlies the system. It needs access only to the system's input-output responses. Then an adaptive fuzzy system encodes this pattern of optimum noise in its if-then rules when gradient learning tunes its parameters. The fuzzy system learns this optimum noise level as it varies the output of a random noise generator. More complex fuzzy systems can themselves act as adaptive random number generators [9].

4.1 SR IN THE QUARTIC BISTABLE SYSTEM

We tested the quartic bistable system $\dot{x} = ax - bx^3 + s(t) + n(t)$ because of its wide use in the SR literature

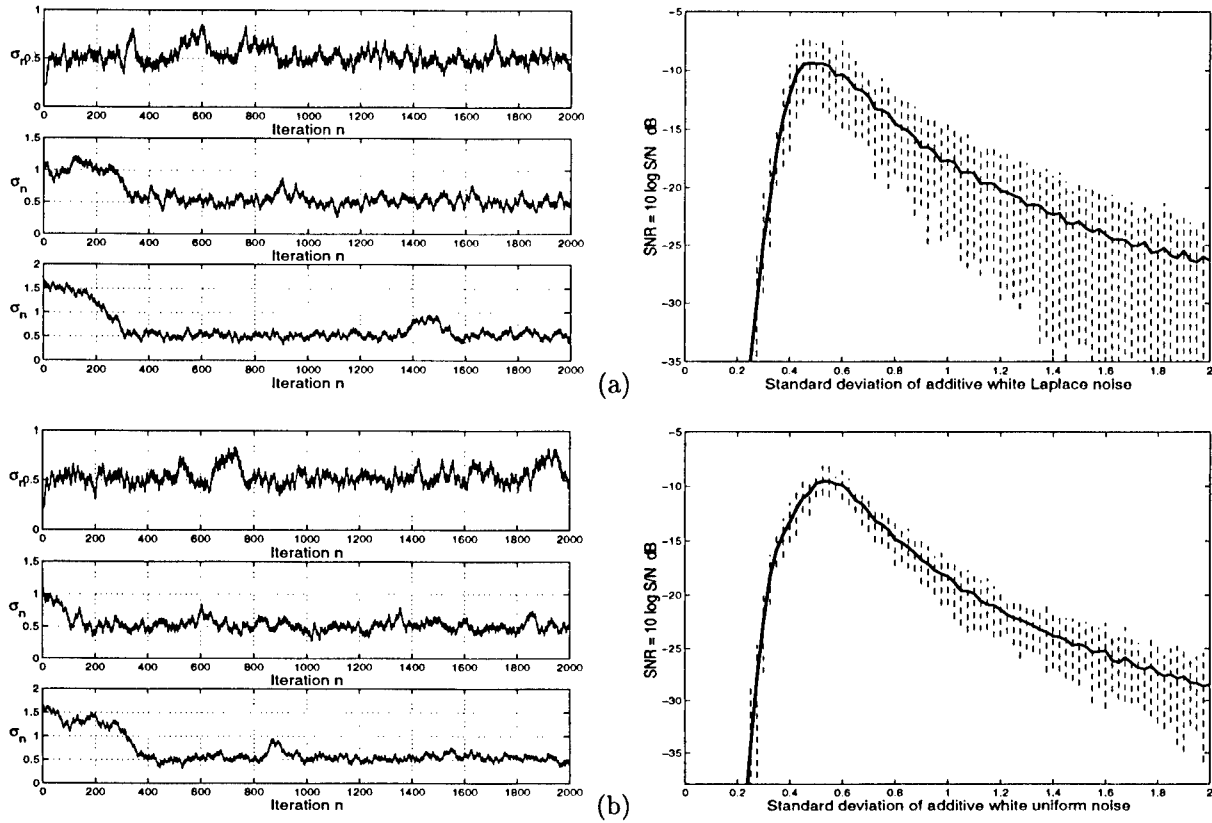


Figure 4: Learning paths of σ_n with the suppressor ϕ for other noise densities in the quartic bistable system (30)-(31) with input signal $s(t) = 0.1 \sin 2\pi(0.01)t$. The noise n has densities (a) Laplace noise and (b) uniform noise. The SNR-noise profiles show that optimal noise levels lie near $\sigma = 0.5$ for both cases.

as a benchmark SR dynamical system. The constants $a = b = 1$ and the binary output give the system [16]

$$\dot{x} = x - x^3 + s(t) + n(t) \quad (28)$$

$$y(t) = \text{sgn}(x(t)) \quad (29)$$

where $s = \varepsilon \sin \omega_0 t$ is a sinewave input forcing term and n is a zero-mean additive white noise with variance $D = \sigma_n^2$. The simulation uses the discrete version:

$$x_{t+1} = x_t + T(x_t - x_t^3 + \varepsilon \sin 2\pi f_0 T t) + \sqrt{T} n_t \quad (30)$$

$$y_t = \text{sgn}(x_t) \quad (31)$$

with initial condition x_0 and time step T . The zero-mean white noise sequence $\{n_t\}$ has variance $D_t = \sigma_n^2(t)$. The term \sqrt{T} scales n_t so that it conforms with the Wiener increment [6]. The simulations use Gaussian noise, Laplace noise, and uniform noise.

We look at the equilibrium term or the random optimality “error” process

$$\mathcal{E} = \frac{S}{N} - \frac{\partial S / \partial \sigma}{\partial N / \partial \sigma} \quad (32)$$

near the optimum noise $\sigma = \sigma_{opt}$. The probability density of \mathcal{E} depends on the statistics of the input

noise, the differential equation that defines the dynamical system, and how we define the signal and noise terms S and N . The empirical test of \mathcal{E}_n found that \mathcal{E}_n had infinite variance in our simulations. The log-tail test of parameter α in the family of alpha-stable probability densities leads to the estimate $\alpha \approx 1.0$. So the \mathcal{E}_n density is approximately Cauchy. Recall also that $Z = X/Y$ is a Cauchy random variable if X and Y are Gaussian or if they obey certain more general statistical conditions [10]. This suggests that much of the impulsive nature of \mathcal{E}_n and hence of the learning process may stem from the ratio of derivatives in (32).

We sample S_n and N_n after a long period of time in (17) and (18). This approximation lets us choose the time length between step n and step $n + 1$. Longer time lengths can better show how the noise intensity σ_n affects S_n , N_n , and the SNR_n . We chose the time length $T_{n+1} - T_n = 2000$ seconds for the simulations. The learning process’s sampling interval T_s differs from the time step T of the dynamical system’s simulator in (30)-(31). The time step is $T = 0.0195$. The sampling period is $T_s = 0.976$ seconds. This yields 2048 samples per iteration. This long period of time allows

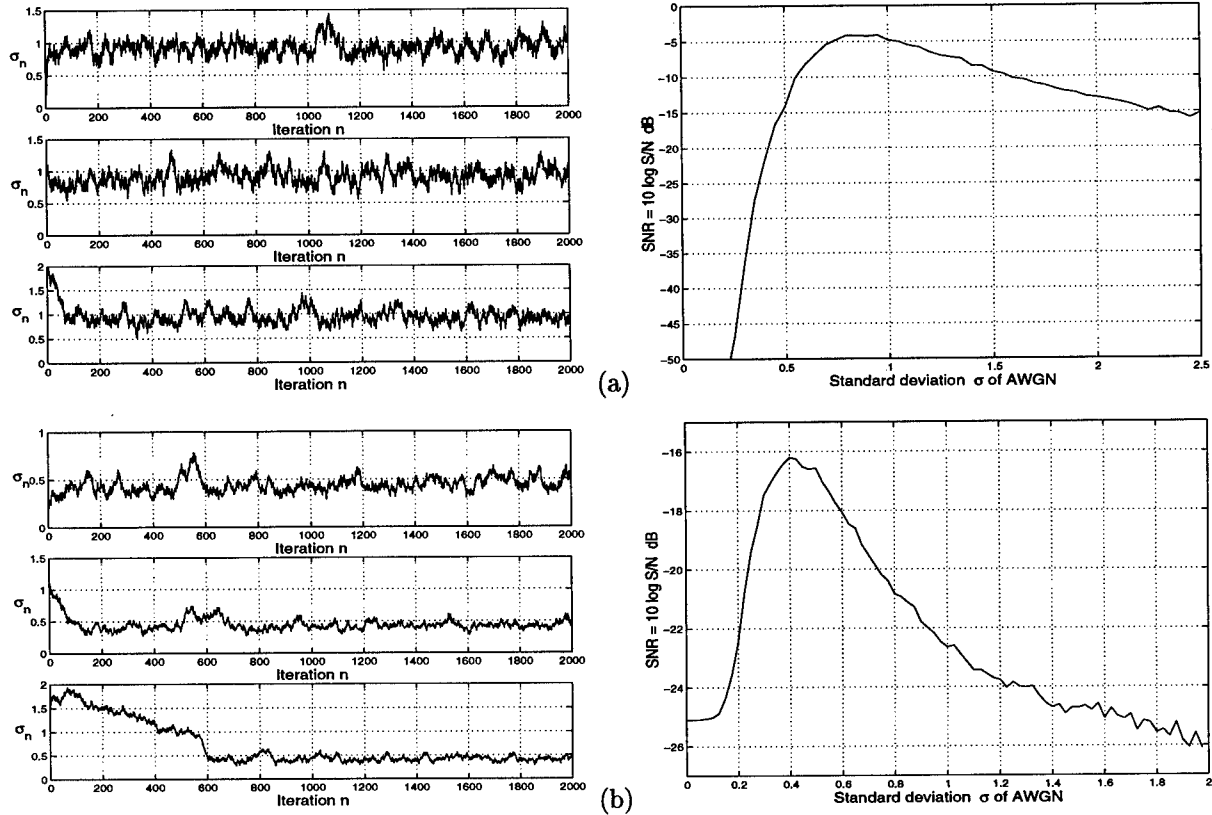


Figure 5: SR learning paths of σ_n for other dynamical systems. (a) The forced bistable neuron model $\dot{x} = -x + 2 \tanh x + \varepsilon \sin(\omega_0 t) + n(t)$ with binary output $y(t) = \text{sgn}(x(t))$. The parameters of the input sinewave are $\omega_0 = 2\pi f_0$ with $f_0 = 0.01$ Hz and $\varepsilon = 0.3$. (b) The FitzHugh-Nagumo neuron model $\dot{\epsilon}x = -x(x^2 - \frac{1}{4}) - w + A + s(t) + n(t)$ and $\dot{w} = x - w$ with output $y(t) = x(t)$. The parameters are $\epsilon = 0.005$ and $A = -(5/12\sqrt{3} + 0.07)$. The sinewave input signal is $s(t) = \varepsilon \sin 2\pi f_0 t$ where $\varepsilon = 0.01$ and $f_0 = 0.5$ Hz.

for low frequency signals such as $f_0 = 0.001$ Hz. We ignored all aliasing effects. We also replace the difference $\sigma_n - \sigma_{n-1}$ with $\text{sgn}(\sigma_n - \sigma_{n-1})$ to avoid numerical instability. The gradient becomes

$$\frac{\partial \text{SNR}_n}{\partial \sigma_n} \approx \left(\frac{\Delta S_n}{S_n} - \frac{\Delta N_n}{N_n} \right) \text{sgn}(\sigma_n - \sigma_{n-1}) \quad (33)$$

for $\Delta S_n = S_n - S_{n-1}$ and $\Delta N_n = N_n - N_{n-1}$. This approximation gives the SR learning law when $F = \sigma_n$:

$$\sigma_{n+1} = \sigma_n + \mu_n \left(\frac{\Delta S_n}{S_n} - \frac{\Delta N_n}{N_n} \right) \text{sgn}(\sigma_n - \sigma_{n-1}). \quad (34)$$

Figures 3a shows sample learning paths of σ_n for the quartic bistable system. The σ_n learning paths converge to the optimum noise values only in some cases. The simulations confirm that the random gradient $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$ in (33) is often impulsive and can destabilize the learning process (34). The impulsiveness of $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$ suggests that it may have an alpha-stable probability density function with parameter $\alpha < 2$. A log-tail test found that $\alpha \approx 1$. This means that $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$ has an approximate Cauchy distribution.

The theory of robust statistics [8] suggests one way to reduce the impulsiveness of $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$. We can replace the noisy random sample z_n with a Cauchy-like noise suppressor $\phi(z_n)$ [8]:

$$\phi(z_n) = \frac{2z_n}{1 + z_n^2}. \quad (35)$$

So $\phi(\frac{\partial \text{SNR}_n}{\partial \sigma_n})$ replaces the approximation of the noise gradient $\frac{\partial \text{SNR}_n}{\partial \sigma_n}$ in (33). This gives the robust SR learning law

$$\sigma_{n+1} = \sigma_n + \mu_n \phi\left(\frac{\partial \text{SNR}_n}{\partial \sigma_n}\right). \quad (36)$$

Figure 3b shows the results of the SR learning law (36) with the gradient in (33). The σ_n learning paths converge to the optimum noise level if the initial value lies close enough to it and then σ_n wanders in a small Brownian-like motion about the optimum noise level.

Like results hold for other noise densities with finite variance such as Laplace and uniform noise. Figure 4

shows σ_n learning paths for the quartic bistable system (30)-(31) with Laplace noise and uniform noise.

4.2 SR IN OTHER DYNAMICAL SYSTEMS

We tested the bistable potential neuron model with Gaussian white noise [4]

$$\dot{x} = -x + 2 \tanh x + s(t) + n(t) \quad (37)$$

$$y(t) = \text{sgn}(x(t)). \quad (38)$$

Figure 5a shows the SR learning paths of σ_n . The sinewave input is $s(t) = \varepsilon \sin 2\pi f_0 t$ where $f_0 = 0.01$ Hz and $\varepsilon = 0.1$ and the $\varepsilon = 0.3$. The time step in the discrete simulation is $T = 0.0195$. The sampling interval is $T_s = 0.975$ or 50 times the time step T .

We next tested the forced FitzHugh-Nagumo neuron model [5, 7, 15]

$$\varepsilon \dot{x} = -x(x^2 - \frac{1}{4}) - w + A + s(t) + n(t) \quad (39)$$

$$\dot{w} = x - w \quad (40)$$

$$y(t) = x(t). \quad (41)$$

The constants are $\varepsilon = 0.005$, $a = 0.5$, and $A = -(5/12\sqrt{3} + 0.07)$. The sinewave input is $s(t) = \varepsilon \sin 2\pi f_0 t$ with $\varepsilon = 0.01$, $f_0 = 0.1$, and 0.5 Hz. The sampling interval is $T_s = 0.01$ with $T = 0.001$. Figure 5b shows the learning paths of the standard deviation σ_n of the Gaussian white noise n .

4.3 FUZZY SR LEARNING: THE QUARTIC BISTABLE SYSTEM

We used a fuzzy function approximator $F : R^n \rightarrow R$ to learn and store the entire surface of optimal noise values for the quartic bistable system with input sinewaves. The fuzzy system had as its input the 2-D vector of sinewave amplitude ε and frequency f_0 . We tested the system with the fixed input initial value $x(0) = -1$. The fuzzy system itself defined a vector function $F : R^2 \rightarrow R$ and used 200 rules. The Cauchy noise suppressor gives the learning law (21) as

$$m_j(n+1) = m_j(n) + \mu_n \phi\left(\frac{\partial \text{SNR}_n}{\partial \sigma_n}\right) \frac{\partial F}{\partial m_j}. \quad (42)$$

Figure 6 shows how we formed a first set of rules on the product space of the two variables ε and f_0 . It also shows how the learning laws move and shape the width of the if-part sinc set. Figure 7 shows the results of SAM learning of the optimal noise pattern for the quartic bistable system. The sinc SAM used 200 rules. Fewer rules gave a coarser approximation.

5 CONCLUSIONS

Stochastic gradient ascent can learn to find the SR mode of at least some simple dynamical systems. This

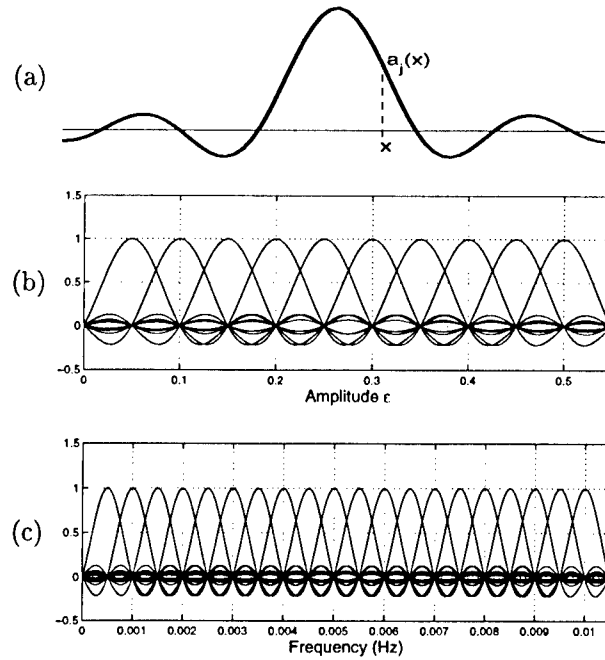


Figure 6: If-part sinc fuzzy sets. (a) Scalar sinc set function $a_j(x) = \sin x/x$. Sinc sets are generalized fuzzy sets with “membership values” in $[-.217, 1]$. Element x belongs to set A_j to degree $a_j(x)$: $\text{Degree}(x \in A_j) = a_j(x)$. (b)-(c) Initial subsets for sinewave amplitudes and frequencies. There are 10 fuzzy sets for amplitude ε and 20 fuzzy sets for frequency f_0 . The product of two 1-D sets gives the 2-D joint sets: $a_j(x) = a_j(\varepsilon, f_0) = a_j^1(\varepsilon) a_j^2(f_0)$. So the product space gives $10 \times 20 = 200$ if-part sets in the if-then rules.

learning scheme may fail to scale up for more complex nonlinear dynamical systems of higher dimension or may get stuck in the local maxima of multimodal SNR profiles. Simulations showed that the key learning term itself can give rise to strong impulsive shocks in the learning process. These shocks often approached Cauchy noise in intensity. A Cauchy noise suppressor gave a working SR learning scheme for the DFT-based SNR measure. Other SNR measures or other process statistics may favor other types of robust noise suppressors or may favor still other techniques to lessen the impulsiveness.

Gradient-ascent learning can find the SR mode of the main known dynamical models that show the SR effect and can do so in the presence of a wide range of noise types. This suggests that SR may occur in many multivariable dynamical systems in science and engineering and that simple learning schemes can sometimes measure or approximate this behavior. We lack formal results that describe when and how such SR learning algorithms will converge for which types of SR systems. This reflects the general lack of a formal

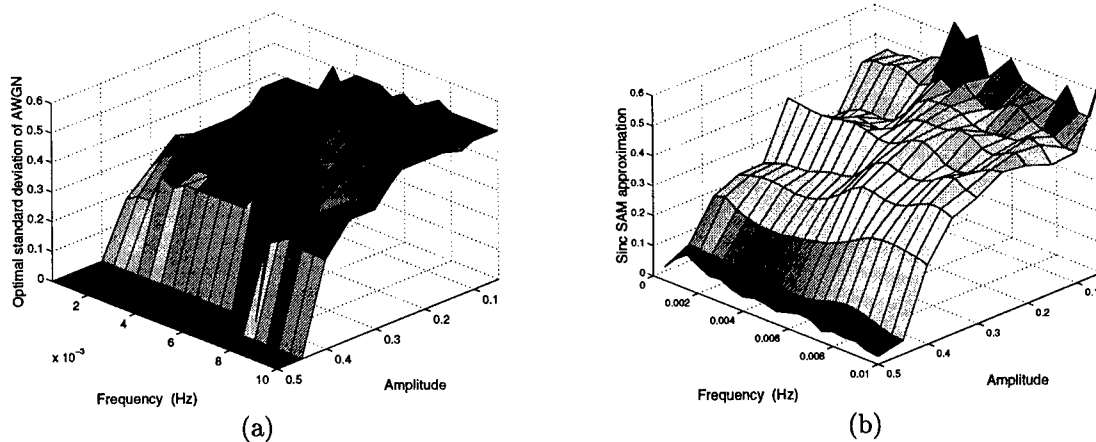


Figure 7: Optimal noise levels in terms of the signal-to-noise ratio for the quartic bistable system (30)–(31). (a) The optimum noise pattern when inputs are sinewaves with distinct amplitudes and frequencies. (b) SAM fuzzy approximation of the optimum noise after 30 epochs. The sinc SAM used 200 rules. One epoch used 20 iterations that trained on 200 input amplitudes and frequencies. The initialized SAM gave the output value 0.2 as its first estimate of the optimal noise level.

taxonomy in this promising new field: Which noisy dynamical systems show what SR effects for which forcing signals?

References

- [1] R. Benzi, G. Parisi, A. Sutera, and A. Vulpiani, "Stochastic Resonance in Climatic Change," *Tellus*, vol. 34, pp. 10–16, 1982.
- [2] R. Benzi, A. Sutera, and A. Vulpiani, "The Mechanism of Stochastic Resonance," *Journal of Physics A: Mathematical and General*, vol. 14, pp. L453–L457, 1981.
- [3] A. R. Bulsara and L. Gammaitoni, "Tuning in to Noise," *Physics Today*, pp. 39–45, March 1996.
- [4] A. R. Bulsara, E. W. Jacobs, T. Zhou, F. Moss, and L. Kiss, "Stochastic Resonance in a Single Neuron Model: Theory and Analog Simulation," *Journal of Theoretical Biology*, vol. 152, pp. 531–555, 1991.
- [5] J. J. Collins, C. C. Chow, and T. T. Imhoff, "Stochastic resonance without tuning," *Nature*, vol. 376, pp. 236–238, July 1995.
- [6] J. L. Doob, *Stochastic Processes*, John Wiley & Sons, 1953.
- [7] C. Heneghan, C. C. Chow, J. J. Collins, T. T. Imhoff, S. B. Lowen, and M. C. Teich, "Information Measures Quantifying Aperiodic Stochastic Resonance," *Physical Review E*, vol. 54, no. 3, pp. R2228–R2231, 1996.
- [8] P. J. Huber, *Robust Statistics*, John Wiley & Sons, 1981.
- [9] B. Kosko, *Fuzzy Engineering*, Prentice Hall, 1996.
- [10] R. G. Laha, "On a Class of Distribution Functions Where the Quotients Follows the Cauchy Law," *Transactions of the American Mathematical Society*, vol. 93, pp. 205–215, November 1959.
- [11] J. E. Levin and J. P. Miller, "Broadband Neural Encoding in the Cricket Cercal Sensory System Enhanced by Stochastic Resonance," *Nature*, vol. 380, pp. 165–168, March 1996.
- [12] S. P. Lipshitz, R. A. Wannamaker, and J. Vanderkooy, "Quantization and Dither: A Theoretical Survey," *Journal of Audio Engineering Society*, vol. 40, no. 5, pp. 355–374, May 1992.
- [13] S. Mitaim and B. Kosko, "What is the Best Shape for a Fuzzy Set in Function Approximation?," in *Proceedings of the 5th IEEE International Conference on Fuzzy Systems*, September 1996, vol. 2, pp. 1237–1243.
- [14] R. P. Morse and E. F. Evans, "Enhancement of Vowel Coding for Cochlear Implants by Addition of Noise," *Nature Medicine*, vol. 2, pp. 928–932, August 1996.
- [15] F. Moss, J. K. Douglass, L. Wilkens, D. Pierson, and E. Pantazelou, "Stochastic Resonance in an Electronic FitzHugh-Nagumo Model," in *Annals of the New York Academy of Sciences Volume 706: Stochastic Processes in Astrophysics*, J. R. Buchler and H. E. Kandrup, Eds., 1993, pp. 26–41.
- [16] F. Moss, D. Pierson, and D. O'Gorman, "Stochastic Resonance: Tutorial and Update," *International Journal of Bifurcation and Chaos*, vol. 4, no. 6, pp. 1383–1397, 1994.
- [17] C. Nicolis and G. Nicolis, "Stochastic Aspects of Climate Transitions—Additive Fluctuations," *Tellus*, vol. 33, pp. 225–234, 1981.
- [18] K. Ogata, *Modern Control Engineering*, Prentice-Hall, third edition, 1997.
- [19] K. Wiesenfeld and F. Moss, "Stochastic Resonance and the Benefits of Noise: From Ice Ages to Crayfish and SQUIDS," *Nature*, vol. 373, pp. 33–36, January 1995.

Q2: Memory-based active learning for optimizing noisy continuous functions

Andrew W. Moore¹², Jeff G. Schneider¹², Justin A. Boyan¹, and Mary S. Lee²

CMU Computer Science & Robotics¹ Schenley Park Research Inc.²

Pittsburgh, PA 15213

6413 Howe Street

<http://www.cs.cmu.edu/~AUTON>

Pittsburgh, PA 15206

{awm,schneide,jab,mslee}@cs.cmu.edu

Abstract

This paper introduces a new algorithm, Q2, for optimizing the expected output of a multi-input noisy continuous function. Q2 is designed to need only a few experiments, it avoids strong assumptions on the form of the function, and it is autonomous in that it requires little problem-specific tweaking.

These capabilities are directly applicable to industrial processes, and may become increasingly valuable elsewhere as the machine learning field expands beyond prediction and function identification, and into embedded active learning subsystems in robots, vehicles and consumer products.

Four existing approaches to this problem (response surface methods, numerical optimization, supervised learning, and evolutionary methods) all have inadequacies when the requirement of “black box” behavior is combined with the need for few experiments. Q2 uses instance-based determination of a convex region of interest for performing experiments. In conventional instance-based approaches to learning, a neighborhood was defined by proximity to a query point. In contrast, Q2 defines the neighborhood by a new geometric procedure that captures the size and shape of the zone of possible optimum locations. Q2 also optimizes weighted combinations of outputs, and finds inputs to produce target outputs.

We compare Q2 with other optimizers of noisy functions on several problems, including a simulated noisy process with both non-linear continuous dynamics and discrete-event queueing components. Results are encouraging in terms of both speed and autonomy.

1 ACTIVE LEARNING FOR OPTIMIZATION

The apparently humble task of parameter tweaking for noisy systems is of great importance whether the parameters being tweaked are for an algorithm, a real manufacturing process, a simulation, or a scientific experiment. The purpose of this paper is two-fold. First, we wish to highlight the potential importance of machine learning as an as-yet underexploited tool in this domain. Second, we will introduce Q2, a new algorithm designed for this domain.

We consider a generalized noisy optimization task in which a vector \mathbf{x} of real-valued inputs produces a scalar output y that is a noisy function of \mathbf{x} :

$$y = g(\mathbf{x}) + \text{noise} \quad (1)$$

Given a constrained space of legal inputs, the task is to find the input vector \mathbf{x}_{opt} that maximizes g , using only a small number of experiments.

In both industrial settings and in algorithm-tuning, this task often demands considerable human intervention and insight. A factory manager who wants to optimize a process can:

- Buy a computer, statistics software, and hire a professional statistician to solve the problem using insight and experiment design.
- Save money and try to “wing it” by manually tuning the parameters.

For highly expensive or safety-critical processes, the first option is always preferable, leaving only the question of which are the best analysis and experiment design tools for the statistician to use. This area is heavily investigated by the academic statistics community.

But there are also many situations in which it is impractical to enlist human-aided analysis during optimization, for example if a vehicle engine self-tunes

during driving. And there are many other situations in which the potential benefit from optimization is too small to justify paying for expert professional analysis. In such cases, it is tempting to ask: Can "black box" automated methods optimize noisy systems? If practical black box methods are found, they could be widely used. Somewhat fancifully, this could lead to the eventual inclusion of Black Box Optimizer chips within a huge range of consumer products, from vehicle engines and industrial equipment down to refrigerators, toasters, and toys.

In the next section we discuss variants of the Black Box Noisy Optimization task. Then in Section 3 we discuss existing approaches. After that we present and evaluate Q2, a new algorithm.

2 VARIANTS OF NOISY OPTIMIZATION

The generalized noisy optimization task summarized by Equation 1 has many variants. For instance, in some domains each experiment is a lengthy procedure, and so there is ample computation time between experiments. In other domains, experiments are very quick, leaving an optimizer little time to make its recommendations. The specifics of the domain determine which methods are appropriate. The following factors need to be considered:

- **Minimize regret or the number of experiments?** Do we pay a constant cost per experiment, or do experiments with poor results cost us more? In scenarios such as tuning the parameters for an algorithm, or optimizing a test plant in which all products will be discarded, the cost per experiment may be constant. But in a task such as minimizing the fuel consumption of a running engine, some experiments cost more than others. Here, we focus on simply minimizing the number of experiments. Note that this presumes that we are not risk-averse: there is no penalty for performing highly unpredictable experiments.
- **How much computer time is available to choose experiments?** If experiments are very cheap and very quick, then an algorithm that needs extensive CPU time to select the ideal next experiment could still be inferior to one that requires only a fraction of a second to suggest a reasonable-but-less-than-ideal experiment. Here, we assume that experiments are costly enough (in time or money) that it pays to choose them carefully. But the Q2 algorithm can be adjusted to satisfy any desired tradeoff between the speed and the quality of proposed experiments.
- **Are we doing local or global optimization?** Unless we have strong prior knowledge, global optimization of a function of more than a couple of inputs requires a very large number of experiments. Q2 is only designed to find a local optimum, though empirically it appears to be good at discovering the global optimum.
- **Can we re-use old data?** Many algorithms have a "current location" or "current set of k recent evaluations" but otherwise disregard earlier evaluations. Q2, however, can exploit any existing data, including previous evaluations obtained by other experimental methods.

In this paper we also assume that there are no long term dynamics, i.e. the output of the n 'th experiment depends only on the n 'th chosen \mathbf{x} , not on previous \mathbf{x} values or the time. Unlike [2, 6] we only try to find the optimum, not to model the g function.

3 POSSIBLE APPROACHES

Many disciplines have methods that are relevant to noisy optimization. Space permits only a brief survey.

Numerical analysis: Numerical methods such as Newton-Raphson or Levenberg-Marquardt [11] have fast convergence properties, but they must be applied carefully to prevent oscillations or divergence to infinity, which violates our desire for black box autonomy. Furthermore, current numerical methods cannot survive noise.

Stochastic approximation: The algorithm of [12] finds roots without the use of derivative estimates. Keifer-Wolfowitz (KW) [5] is a related algorithm for noisy optimization. It estimates the gradient by performing experiments in both directions along each dimension of the input space. Based on the estimate, it moves its experiment center and repeats. It uses decreasing step sizes to ensure convergence. KW's strengths are its aggressive exploration, its simplicity, and that it comes with convergence guarantees. However, it can attempt wild experiments if there is noise, and discards the data it collects after each gradient estimate is made. Amoeba (see below) is a similar approach, but in our experience is superior to KW.

Amoeba search: Amoeba [11] searches k -d space using a simplex (i.e. a k -dimensional tetrahedron). The function is evaluated at each vertex. The worst-performing vertex is reflected through the hyperplane defined by the remaining vertices to produce a new simplex that has moved up the estimated gradient. Ingenious simplex transformations let the simplex shrink near the optimum, grow in large linear zones, and ooze along ridges.

Experiment design & response surface methods: Current RSM practice is described in the classic reference [1]. It proceeds by cautious steepest ascent hill-climbing. A region of interest (ROI) is established at a starting point and experiments are made at positions that can best be used to identify local function properties with low-order polynomial regression. Much of the RSM literature concerns *experimental design*—deciding where to take data in order to acquire the lowest variance estimate of the polynomial coefficients in a fixed number of experiments. When the gradient is estimated confidently, the ROI is moved accordingly. Quadratic regression locates optima within the ROI, and diagnoses ridge systems and saddle points. The strength of RSM is that it avoids changing operating conditions based on inadequate evidence, but moves once the data justifies it. A weakness of RSM is that human judgment is needed: it is not an algorithm, but a manufacturing methodology.

Evolutionary computation and learning automata: Methods such as genetic algorithms begin by sampling uniformly, but then bias later samples in favor of the experiments that had good outcomes. There is a vast literature of refinements of such methods. These approaches need thousands, sometimes millions, of evaluations, because they attack a different problem: Global Optimization, usually for noise-free, cheap-to-evaluate criteria.

PMAX: PMAX is a simple, effective algorithm. Based on the data from the experiments so far, it uses a non-linear function approximator to estimate the underlying function $g(\mathbf{x})$. The next experiment is taken at the point that maximizes the estimate of g . This approach has been used with a decision-tree approximator [13], with neural nets (in many commercial products), and with locally weighted regression [9]. Variations of PMAX include taking the next experiment not at the predicted optimum, but instead where the confidence intervals are widest [6], or where the top of the confidence interval is maximized [9], or in accordance with the Interval Estimation heuristic [4] or similar criteria [13].

Empirically, we have found that PMAX using locally weighted regression as the function approximator is often faster than more sophisticated alternatives [9]. However it has some serious drawbacks:

- In conventional function approximation one must solve the bias-variance tradeoff. This is often determined automatically using cross-validation [8], but this proves difficult with a set of very few, weirdly distributed datapoints obtained during optimization. Empirically we have observed dismal performance when attempting this. In addition, conventional approaches search for the best model over the whole data range, whereas we only

need our model to be accurate in the vicinity of the optimum.

- PMAX is very expensive. It needs to train a function approximator each time an experiment is made, and then the approximate function must be numerically optimized to produce the suggested experiment.
- PMAX can get stuck in hallucinated optima since it is not choosing experiments to give the most information (in the way that RSM does).

4 THE Q2 ALGORITHM

The Q2 algorithm is an attempt to combine the strengths of Newton's method (superlinear convergence), RSM (using estimates of significance in the face of noise), and PMAX (exploiting all available data). Let us first outline the structure of the Q2 algorithm, before discussing its details:

1. Input a set of previous experimental results

$$(\mathbf{x}_1 \rightarrow y_1), (\mathbf{x}_2 \rightarrow y_2), \dots, (\mathbf{x}_n \rightarrow y_n) \quad (2)$$

and *HR*: a hyper-rectangular portion of input space over which the optimization is constrained to take place.

2. Select a convex Region Of Interest (ROI) within HR such that:

- The constrained optimum within HR is expected to lie within ROI.
- There is no evidence to contradict the assumption that the function is well-approximated by a quadratic within ROI.

3. Select a useful experiment to take within ROI.

4. Return the experiment, the estimated location of the optimum, and (optionally) other information such as the ROI and a regression analysis of the local quadratic.

In typical operation, the suggested experiment will be performed, we will add the new datapoint to the dataset, and return to Step 2.

Step 2: Selecting the ROI

Step 2 begins by generating a sequence of candidate Regions Of Interest, $ROI_1, ROI_2, \dots, ROI_j, \dots$ from which the final ROI will be selected. The generated sequence has the properties that

$$ROI_1 := HR \text{ and } ROI_j \supseteq ROI_{j+1} \quad (3)$$

where ROI_{j+1} is determined by cutting away an unpromising subregion of ROI_j . How is the cut determined? Let us consider an example.

Figure 1 shows a Gaussian function of two inputs. Suppose HR is set to be the full square region depicted in the figure, and suppose we have available the thirty noisy datapoints that are also shown. Call this dataset DS_1 . We can fit a quadratic to DS_1 . Write

$$\hat{y}_k = c + \mathbf{b}^T \mathbf{x}_k + \frac{1}{2} \mathbf{x}_k^T \mathbf{A} \mathbf{x}_k \quad (4)$$

where A is symmetric, or, equivalently,

$$\hat{y}_k = c + b_1 x_{k1} + b_2 x_{k2} + \frac{1}{2} a_{11} x_{k1}^2 + a_{12} x_{k1} x_{k2} + \frac{1}{2} a_{22} x_{k2}^2 \quad (5)$$

The regression is a matter of simple matrix manipulation. Write \mathbf{z}_k = the vector of polynomial terms for the k th input point, \mathbf{x}_k .

$$\mathbf{z}_k = (1, x_{k1}, x_{k2}, x_{k1}^2, x_{k1}x_{k2}, x_{k2}^2) \quad (6)$$

Write \mathbf{Z} = a matrix whose k th row is \mathbf{z}_k , and write \mathbf{Y} = a vector whose k th element is y_k . Finally define

$$\boldsymbol{\beta} = (c, b_1, b_2, \frac{1}{2}a_{11}, a_{12}, \frac{1}{2}a_{22})^T \quad (7)$$

as the regressed coefficients. Then using Bayesian regression with non-informative priors on $\boldsymbol{\beta}$ and σ^2 (the estimated Gaussian noise), we have the MAP of $\boldsymbol{\beta}$ (also the maximum likelihood value in this case) as

$$\boldsymbol{\beta} = (\mathbf{Z}^T \mathbf{Z})^{-1} \mathbf{Z}^T \mathbf{Y} \quad (8)$$

In practice, if the information is known, we can put Gaussian priors on the coefficients and an inverse-Gamma prior on the noise. For our dataset the resulting quadratic approximation is shown in Figure 2. Note that because the underlying function is so far from quadratic, this is a poor fit.

Q2 evaluates each of the datapoints in DS_1 using the quadratic, producing the values of Equation 5. Let $(\mathbf{x}_{k(1)}, y_{k(1)})$ be the datapoint that is predicted to be the worst, i.e. $k(1) = \arg\min_k \hat{y}_k$. It will be used to define a cut of ROI_1 . We look at the direction of the steepest gradient, $\nabla \hat{y}$, of the quadratic at $\mathbf{x}_{k(1)}$, and we cut using the half-plane perpendicular to this direction so that

$$ROI_2 = ROI_1 \cap \{\mathbf{x} \mid (\mathbf{x} - \mathbf{x}_{k(1)}) \cdot \mathbf{d}_1 \geq 0\} \quad (9)$$

where $\mathbf{d}_1 = \nabla \hat{y}$ evaluated at $\mathbf{x}_{k(1)}$.

In Figure 2, the worst point according to the quadratic is at the top left, and with some effort the resulting cut-plane can be seen.

Why do we use the above approach? We want to use our unreliable (probably biased) quadratic to tell us how to reduce the ROI . We assume that even if the quadratic is a poor model for g , it will be adequate to predict an *unpromising* location for the optimum. Why pick the point with the *predicted* worst value instead of the actual worst value? Because the actual values are noisy, meaning that an unlucky datapoint could be misleadingly removed.

We have described how ROI_2 is constructed from ROI_1 . In general, ROI_{j+1} is constructed from ROI_j using a similar recipe: set $DS_{j+1} = DS_j - (\mathbf{x}_{k(j)}, y_{k(j)})$, do a regression using dataset DS_{j+1} (which will be less biased than using DS_j), and cut using the point that the new regression predicts will be worst. Figure 3 shows the approximation that results after the first cut has been made (giving a less biased fit than Figure 2), and also shows the second cut. Figure 4 shows what remains after the twelfth cut: the fit is now good, because it is only based on datapoints near the quadratic-shaped optimum. Figures 5–7 use a bigger dataset and an extreme ridge system.

At this point Q2 has generated a series of candidate regions, ROI_1, ROI_2, \dots . To decide which to select, we perform regression analysis on the quadratics in each of the ROI s. As j increases, ROI_j shrinks and is based on fewer datapoints. So, as j increases, ROI_j 's bias decreases and its variance increases. We select the ROI_j with the best tradeoff using the criterion: *Choose the smallest ROI for which Bayesian regression analysis is confident about the location of the optimum, and for which the optimum is, with high probability, inside the ROI.*¹

The results of this criterion are shown in Figures 8–13. With fewer or noisier datapoints, larger ROI s are chosen. The shape of the chosen ROI s nicely reflects the shape of the local ridge system (Figure 7). If irrelevant inputs are included, the ROI chosen by Q2 tends to stretch to ignore irrelevant dimensions (pictures omitted because of space constraints).

Step 3: Choosing the experiment

Once the ROI is determined, the estimated optimum is easily obtained as

$$\hat{\mathbf{x}}_{\text{opt}} = -\mathbf{A}^{-1} \mathbf{b} \quad (10)$$

(assuming the quadratic fit has revealed a maximum, meaning \mathbf{A} is negative-definite). $\hat{\mathbf{x}}_{\text{opt}}$ is not necessarily the best place to experiment in order to gain useful new information. Instead, we investigated these options:

1. Put experiment at $\hat{\mathbf{x}}_{\text{opt}}$.
2. Choose a random point within ROI .
3. Choose the point in ROI that is predicted to most reduce the uncertainty about the location of the

¹This is achieved by taking the joint posterior distribution (normal-gamma) on the noise and the coefficients of the quadratic form, and then (via Monte Carlo sampling) seeing whether at least $\tau = 98\%$ of the samples lie in the ROI and whether the expected regret of committing to the optimum is below a threshold (2% of the range of output values). Empirically these threshold choices are not performance-critical.

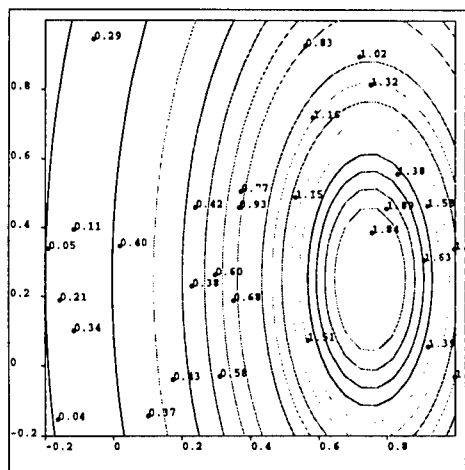


Figure 1: A function of two inputs. The optimum is at (0.75,0.25). It is a Gaussian bump, and hence very flat more than about 0.4 units of distance from the optimum. Also shown are 30 noisy datapoints. These were generated with uniformly random (x,y) coordinates, with z (height) set to $f(x,y)$ plus Gaussian noise with standard deviation 0.1.

optimum.

4. Choose the point in *ROI* that keeps the regression as orthogonal [1] as possible, mimicking established RSM practice.
5. Choose the point in *ROI* as far away from any previous datapoints (in or out of *ROI*) as possible.

Option 5 is best empirically. This is because options 3 and 4, despite their elegance, usually choose experiments at the edge of the *ROI*, reducing the opportunity for future cuts to shrink future *ROIs*. Option 1 quickly becomes stuck, and option 2 frequently wastes experiments.

Details

In this short paper, many details have been omitted. Some regressions predict a minimum or a saddlepoint, instead of a maximum. We have special-purpose techniques to deal with this. The Bayesian analysis is largely standard, and also omitted: see [3] for more details. Some confidence measures require Monte Carlo integration. These details will be discussed in a forthcoming technical report [10].

5 RESULTS

We begin by comparing Q2 with four versions of Amoeba and three versions of PMAX on the function f_1 from Figure 1 with noise of 0.3 added to each evaluation². Amoeba is the classic search algorithm

²These tasks are available from <http://www.cs.cmu.edu/~AUTON>

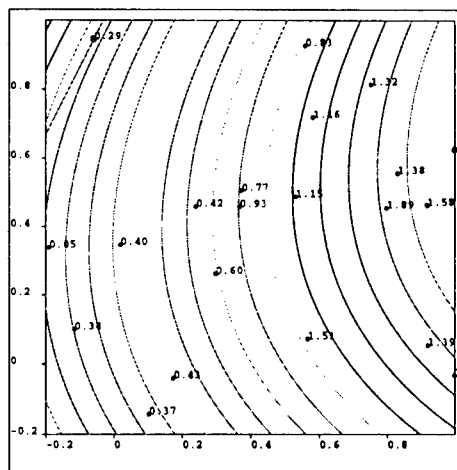


Figure 2: The best-fitting global quadratic regression approximation obtained by least squares regression on the 30 datapoints. The worst-scoring datapoint is in the top left.

from [11]. Amoeba2 is the same except it is made resistant to noise by doing two evaluations and taking their average at each simplex vertex. Amoeba4 and Amoeba8 similarly average four and eight evaluations at each vertex. All the Amoebas begin with a medium-sized simplex started randomly in input space.

The results are in Figure 14. In this (and all subsequent experiments) we performed 25 independent runs of each optimizer, with each run consisting of 60 experiments. As well as selecting the datapoints for the experiments, at every stage the optimizers also gave their estimate of the location of the optimum. To assess the various optimizers, we wish to compare how good they are at estimating the optimum, and so we look at the true value of the underlying function at these estimates of the optimum. For the i th run of a particular optimizer, let s_i denote the mean of the true values at the estimates of the optimum. The figures in the left hand column are the mean s_i value of the optimizer over all 25 runs (i.e. $(\sum_i s_i)/25$). These values are also drawn graphically in the same column: the further to the right the dot lies, the better the mean score. The horizontal lines depict the 95% confidence intervals on the mean. The right hand column shows the mean performance of the optimizer on the final 15 of the 60 experiments. Unsurprisingly, all methods do better in later experiments, so the right hand means are higher.

Figure 14 shows that Q2 outperforms all the other methods on this problem. Amoeba4 is the best of the Amoebas; it is less affected by noise than Amoeba and Amoeba2, but it makes better progress than Amoeba8, which wastes 8 evaluations on every vertex.

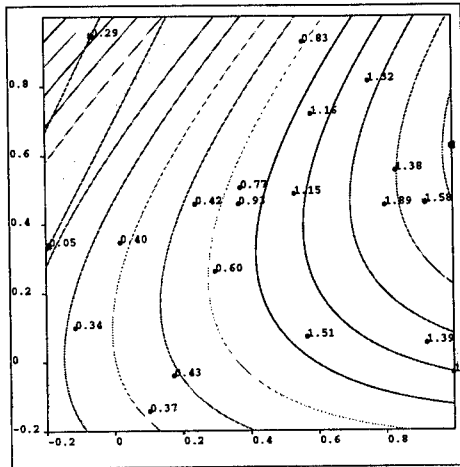


Figure 3: After the worst-scoring point is removed from the regression, we have the following fit to the remaining 29 datapoints. The worst predicted point among these is halfway up along the left edge. Note the cut that it causes.

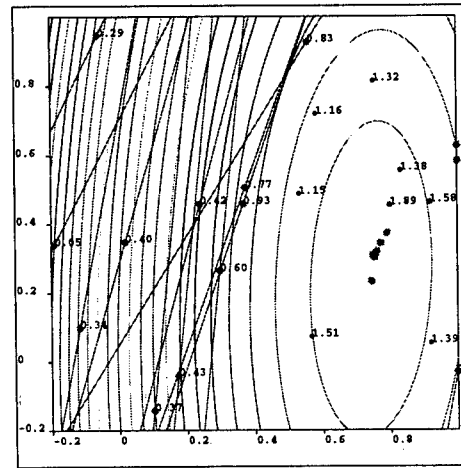


Figure 4: After 12 cuts the remaining datapoints (those inside the convex region defined by the cuts) are relatively close to the optimum, and the resulting local quadratic regression is an excellent local approximation.

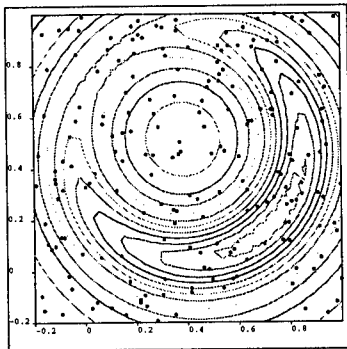


Figure 5: Another function of two inputs. The optimum is on the banana-shaped ridge at (0.75, 0.2). 200 datapoints are shown (their heights omitted).

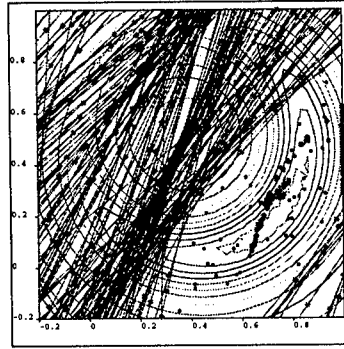


Figure 6: After the first 150 cuts, the region of interest nicely surrounds the ridge.

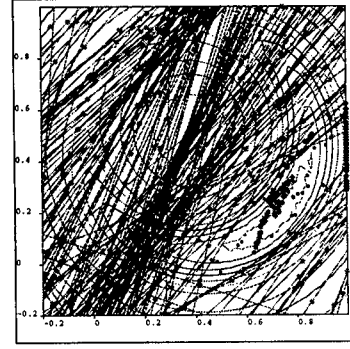


Figure 7: After the first 180 cuts, the region of interest is smaller still, yet continues to surround the true optimum.

Table 1 (shown later) gives results for the 2d-functions of Figures 5, 15, and 16 for noise levels of 0 and 0.3. With no noise, the one-evaluation-per-step version is always the best Amoeba. With noise, the best Amoeba is problem specific. The best PMAX is also problem specific. Q2 adapts well to noise and to differing levels of function complexity. Q2 is beaten by the Global and mediumly local PMAX for the noisy pure quadratic $f_3(x_1, x_2)$. In all other cases Q2 wins, but its main strength is autonomy: unlike Amoeba and PMAX no problem specific parameter needs to be chosen to make Q2 perform well.

Figure 17 shows a simulated, sanitized version of a real industrial process. Liquids enter a tank at a certain rate (a parameter) and a certain mix-ratio (a parameter) unless the tank is above a certain level (a parameter). They react causing a color dependent on the tank mix-ratio and the time spent in the tank. Thickener is

added at a certain rate (a parameter), and the output passes through a cooling tunnel to wait on a holding belt. While waiting, color may change. When the belt fills beyond a certain level (a parameter), production halts. Customer demand randomly consumes material on the holding belt. The yield is the amount of material that reaches the customer with color lying in an acceptable tolerance range. This is a very noisy task. The yield is a highly non-quadratic function; one input is almost irrelevant, the others are all important, and two of the inputs must run to their maximum legal value for best performance. The results are given in Figure 18, and show a significant win for Q2. Q2 and the PMAX's also have far more repeatable results than the Amoebas.

We also applied conventional RSM to this task, using a star design prescribed by [1]. The star occupied the hyperrectangle defined by the legal ranges of values for

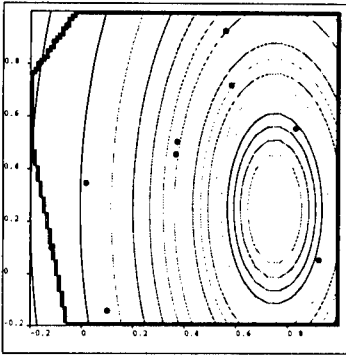


Figure 8: The region of interest selected for the function of Figure 1 given a dataset of only 10 points.

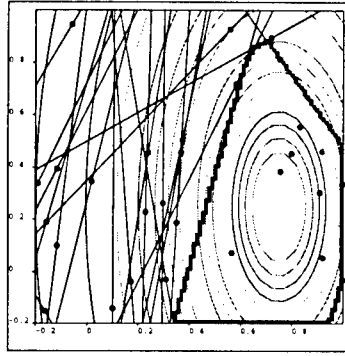


Figure 9: The region of interest when given 30 datapoints.

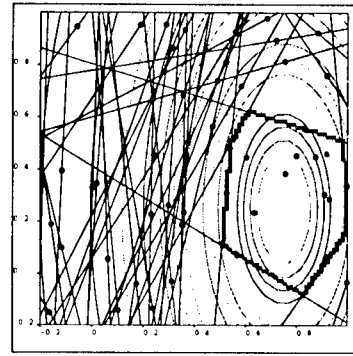


Figure 10: The region of interest when given 50 datapoints.

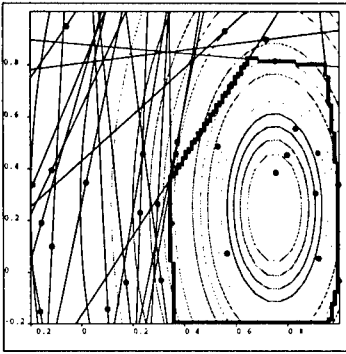


Figure 11: The region of interest selected for the function of Figure 1 given a dataset of 30 points, with no noise.

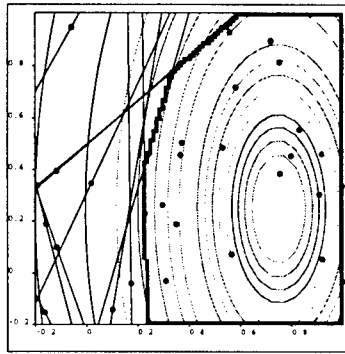


Figure 12: The region of interest when noise with std. dev. $\sigma = 0.5$ is added to the observations.

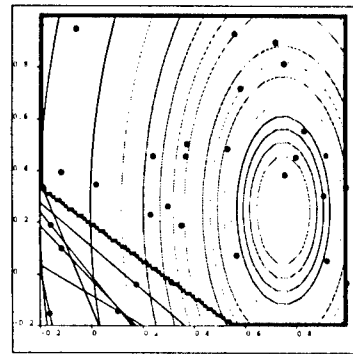


Figure 13: The region of interest when $\sigma = 2.0$.

each input. It needed 76 evaluations, but the chosen optimum had a yield below 10 units: worse than all the other methods, indicating that the assumption of a global quadratic is inadequate in this domain.

Next, we examine a domain where experiments are time-consuming. Figure 19 shows a generalization of the multi-buffer machine task described in [7] (this makes 10 products instead of 5). There are two inputs defining a simple parameterized policy for when to service the machine. Services are costly, but unscheduled breakdown is much worse. This task is evaluated by a computationally expensive simulation; for each setting of the two inputs, we perform 10000 simulation steps to evaluate the performance. Evaluations are very stochastic (with highly non-Gaussian noise). The results are shown for runs of only 24 experiments. Q2 learns a good policy in these 24 experiments, i.e. a total of only 24×10000 simulation steps. This compares favorably with the tens of millions of simulation steps needed for reinforcement learning in [7], but Q2 is unlikely to find as good a policy as their semi-MDP formulation.

The final results show Q2 being used for root-finding

instead of optimization. The hand position in Figure 21 is a noisy function of θ_1 and θ_2 . The task requires us to achieve the goal hand position. Although space permits no details, the version of Q2 for root (or target) finding uses linear instead of quadratic regression in its ROIs. The results are shown in Figure 22. Figure 23 shows the results when, on each experiment, the target position is varied randomly within the workspace. Amoeba, a pure optimization method for a fixed goal, is no longer applicable here, but PMAX and Q2 can still be used because their decision making simply requires a dataset of previous experiences. Q2's ability to tune its regions of interest decisively beats all PMAXs.

	Mean over all 100 trials	Mean over last 25 trials
PmaxGlobal	-0.417 \rightarrow	-0.368 \rightarrow
PmaxLocal	-0.402 \rightarrow	-0.342 \rightarrow
PmaxVLocal	-0.475 \rightarrow	-0.418 \rightarrow
Q2 (Linear)	-0.042 \bullet	-0.021 \bullet

Figure 23: Performance on kinematics when the target varies during each experiment.

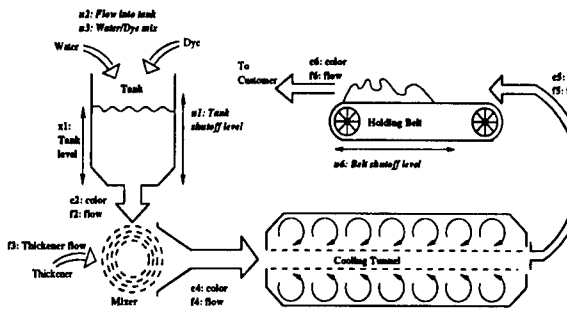


Figure 17: A simulated production process described in the text.

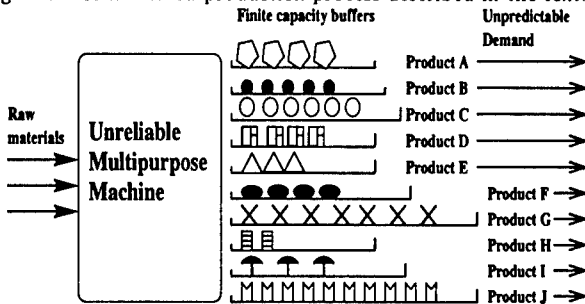


Figure 19: A multi-buffer servicing task similar to those described in [7].

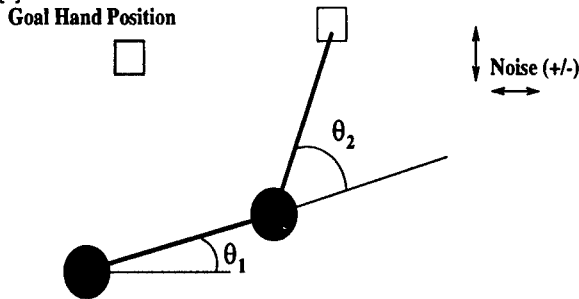


Figure 21: A 2-input, 2-output kinematics task.

	Mean over all 60 trials	Mean over last 15 trials
Amoeba	25.297	27.130
Amoeba2	27.412	33.216
Amoeba4	22.302	26.534
Amoeba8	19.858	21.411
PmaxGlobal	28.006	37.237
PmaxLocal	27.634	37.952
PmaxVLocal	23.012	27.105
Q2	36.334	45.589

Figure 18: Performance on the simulated production process.

	Mean over all 24 trials	Mean over last 6 trials
Amoeba	1.553	1.897
Amoeba2	1.711	2.859
Amoeba4	0.940	2.179
PmaxGlobal	-1.48%	-1.380%
PmaxLocal	-1.52%	-1.454%
PmaxVLocal	-1.56%	-1.51%
Q2	0.535	3.352

Figure 20: Performance at the multi-buffer task.

	Mean over all 40 trials	Mean over last 10 trials
Amoeba	-0.084	-0.082
Amoeba2	-0.092	-0.087
Amoeba4	-0.096	-0.096
Amoeba8	-0.084	-0.071
PmaxGlobal	-0.051	-0.046
PmaxLocal	-0.050	-0.042
PmaxVLocal	-0.057	-0.054
Q2 (Linear)	-0.062	-0.023

Figure 22: Performance on the kinematics task.

- [5] H. Kushner and D. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.
- [6] D. J. C. MacKay. Bayesian Model Comparison and Backprop Nets. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, April 1992.
- [7] S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-Improving Factory Simulation using Continuous-Time Average-Reward Reinforcement Learning. In *Proceedings of the 14th International Conference on Machine Learning (ICML '97)*, Nashville, TN. Morgan Kaufmann, July 1997.
- [8] A. W. Moore, D. J. Hill, and M. P. Johnson. An Empirical Investigation of Brute Force to choose Features, Smoothers and Function Approximators. In S. Hanson, S. Judd, and T. Petsche, editors, *Computational Learning Theory and Natural Learning Systems, Volume 3*. MIT Press, 1992.
- [9] A. W. Moore and J. Schneider. Memory-based Stochastic Optimization. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Neural Information Processing Systems 8*, 1996.
- [10] A. W. Moore, J. Schneider, J. Boyan, and M. S. Lee. Q2: A memory-based active learning algorithm for Blackbox Noisy Optimization. In preparation, 1998.
- [11] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.
- [12] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400-407, 1951.
- [13] M. Salganicoff and L. H. Ungar. Active Exploration and Learning in Real-Valued Spaces using Multi-Armed Bandit Allocation Indices. In *Proceedings of the 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995.

Collaborative Filtering using Weighted Majority Prediction Algorithms

Atsuyoshi Nakamura and Naoki Abe

NEC C& C Media Research Laboratories

4-1-1 Miyazaki, Miyamae-ku, Kawasaki 216-8555 JAPAN

{atsu,abe}@ccm.cl.nec.co.jp

Abstract

We apply various generalizations of weighted majority prediction algorithms for on-line prediction of binary relations to the problem of predicting personal preferences over information contents, which is a key issue in collaborative filtering. Note that the collaborative filtering problem can be casted as learning a binary relation between the users (as the rows) and the contents (as the columns). The original prediction algorithm of Goldman and Warmuth [GW95] makes its prediction by majority voting by the rows with observed data in the same column, weighted by the believed similarity between the rows. In the present paper, we propose a generalization 'G-Learn-Relation' of their algorithm to the multi-valued setting, and empirically demonstrate that it performs better than existing filtering methods based on correlation coefficients, both on simulated and real data. The performance comparison was done in terms of the total number of prediction mistakes and the measures of *precision* and *recall*. Additionally, we propose a version of G-Learn-Relation that makes use of indirect evidence available as believed similarity between other rows, and another version in which both row similarity and column similarity are used for prediction. In both cases, significant improvement was observed in experiments involving simulated data. Finally, we give a theoretical performance guarantee for G-Learn-Relation in terms of an upper bound on the worst case number of mistakes, which together with a lower bound on the number of mistakes made by a correlation-based method establishes that its worst case performance is better than the correlation-based methods.

1 Introduction

We apply various generalizations of weighted majority prediction algorithms, proposed in the context of on-line prediction of binary relations, to the problem of predicting user's preferences on information contents. This is a key issue in personalized information filtering, an area that is gaining increasing attention in internet related technology. Information filtering techniques known in the literature can, for the most part, be classified into two types. One is the contents-based approach to filtering, which is based on the features of the actual contents such as word counts, and the other is the so-called collaborative (or social) filtering approach, which makes use of similarities between the users observed in the past scoring data representing their preferences. Methods combining the two approaches have also been proposed. In this paper, we are concerned with the latter approach, namely filtering methods that are based solely on the scores given by the users on the contents.

Existing methods of collaborative filtering [RISBR94, SM95] make use of correlation coefficients. In this approach, the preference of a user on a particular content is predicted by taking a weighted average of all scores given to that content by various users in the past, weighted by the correlation coefficients between their scores and those of the user in question, calculated using scores given to common contents.¹ These methods are based on a reasonable intuition that correlation coefficients can quantify the similarity between the users' preferences but one shortcoming of this approach is that the estimation confidence of the correlation coefficients is not taken into account.

As a way to address this issue, we resort to on-line prediction algorithms for binary relations proposed and studied in the areas of computational learning theory

¹It has been reported that a variant of this method that uses a threshold and a fixed average do the best among various methods based on the correlation coefficients [SM95].

and machine learning [GRS93, GW95, NA95]. Note that information filtering can in principle be viewed as a learning problem for a binary relation, in which a user is related to a content just in case he or she prefers it. Such a binary relation can be represented by a 0,1-valued matrix, in which the rows represent users and the columns represents contents. In particular, we make use of the weighted majority prediction algorithm proposed and analyzed by Goldman and Warmuth and its generalizations [ALN95]. These methods learn weights that roughly represent the believed similarities between the rows (and columns) and make predictions by weighted majority voting. Here we further extend these algorithms so as to handle the cases in which the scores are not necessarily binary but many-valued.

First, we generalized the original weighted majority prediction algorithm 'Learn-Relation' [GW95] into the many-valued setting. (We call the generalized algorithm 'G-Learn-Relation'.) We evaluated the performance of this method using both simulated and real data. In our evaluation, we considered that a prediction whose round-off integral value is at most one off the correct value to be *correct* and all others to be *mistakes*. The experimental results indicate that G-Learn-Relation out-performs the best known method based on correlation coefficients in experiments using both simulated and real data, in terms of the total number of mistakes. With respect to more widely used measures of *precision* and *recall*, G-Learn-Relation had a better overall performance as well. Furthermore, it was found that G-Learn-Relation is less sensitive to the choice of its parameters, as compared to the correlation-based methods.

Next, we evaluated the effect of using the similarities between the columns as well as the rows in making predictions. It has been verified, using several two-dimensional extensions of Learn-Relation, that such an approach can improve the predictive performance in another application domain [ALN95]. In our experiments using simulated data, the effect of using both rows and columns was observed for both correlation-based methods and for G-Learn-Relation, the two-dimensional extension of G-Learn-Relation being the most favored. With respect to real data, however, the effect was minimal. This may be attributable to the fact that the real data used in our experiments had very uneven number of rows and columns (48 rows and 277 columns).

As an attempt to further improve the performance of G-Learn-Relation, we enhanced its prediction by using indirect evidence. In particular, we incorporate an idea suggested by Lang and Baum [LB97] into the weighted majority prediction algorithm. Their method, which they call 'triple row,' is based on the

idea that 'a friend's friends is a friend, too' (and a friend's enemy is an enemy, too.) That is, in determining the similarity between two rows, we take into account the (dis)similarity between the two rows and a third row. Our experimental results indicate that this enhancement results in a significant performance improvement on simulated data, but on real data the effect was inconclusive.

Finally, we give a theoretical performance guarantee for G-Learn-Relation in terms of an upper bound on the worst case number of mistakes it makes. We also show a lower bound on the worst case number of mistakes made by the correlation-based method and establish that the worst case performance of the weighted majority type algorithms is better than that of the correlation-based methods.

2 The problem formulation

Collaborative filtering using methods that are based solely on the scores given by the users on the contents can be viewed as an on-line prediction problem for binary relations (and multi-valued functions). The target binary relation (or function) can be represented by a matrix M , whose i, j -entry represents the score given by user i on contents j . On-line learning proceeds as follows. At any given time t , the learning algorithm is given an arbitrary pair i, j and predicts its value as M_{ij} , based on an observation matrix O^t . Here an observation matrix O in general satisfies $O_{ij} = M_{ij}$ whenever the i, j entry has been observed, and $O_{ij} = *$ otherwise. The learner is then given the actual value of M_{ij} , and O^t is updated (to O^{t+1}) accordingly. Starting initially with O^0 whose elements are all $*$, the above process is repeated until the matrix is fully observed, namely until $O^t = M$. In our experiments, we assume that the scores are integers between 1 and 5 (5 being the highest score) and prediction is done with a real number. A prediction is considered correct if its round-off value² is at most 1 different from the correct value. The performance of an on-line learning algorithm is measured in terms of the total number of mistakes in the entire trial sequence, often as a function of various parameters quantifying the size of the problem. These include the numbers of rows and columns as well as the numbers of row types and column types, where two rows i, i' are said to belong to the same type, if they agree in all columns, namely if $M_{ij} = M_{i'j}$ holds for all j . (The column types are similarly defined.)

3 Algorithms Employed

In this section, we describe the generalized weighted majority algorithms we propose in this paper.

²For example, the round-off values of 3.4 and 3.5 are 3 and 4, respectively.

The original weighted majority prediction algorithm (Learn-Relation) makes its prediction \hat{M}_{ij} by weighted majority voting by all rows i' such that the entry $M_{i'j}$ in the same column has been observed, each weighted by the weight $w_{ii'}$ representing the believed degree of similarity between the rows i and i' . The weights are updated by multiplying those contributing to the correct value by $(2-\gamma)$ and those contributing to a wrong value by γ , for some $\gamma < 1$. Note that this update is equivalent to defining the weight $w_{ii'}$ at each trial as $(2-\gamma)^{C_{ii'}} 2^{W_{ii'}}$, where $C_{ii'}$ is the number of times i' has voted for a correct value in row i and $W_{ii'}$ the number of times it voted for a wrong value.

G-Learn-Relation generalizes Learn-Relation for multi-valued functions by letting each row i' vote for all values (in $V(a)$) within a permitted tolerance from its predicted value $a = \hat{M}_{ij}$. Its weights are updated in the same manner as in Learn-Relation.

G-Learn-Relation ($0 \leq \gamma < 1$)

With each row pair (i, i') is associated a weight, $w_{ii'}$. We let A denote the range of entries of M , and for any $a \in A$, $V(a)$ denotes the set of prediction values that are considered correct when the true value is a .

Initialization: $w_{ii'} := 1$

Prediction:

$$\hat{M}_{ij} = \begin{cases} \arg \max_{a \in A} \sum_{i': O_{i'j} \in V(a)} w_{ii'} & \text{if } \{i' : O_{i'j} \neq *\} \neq \emptyset \\ C_0 (\text{a constant}) & \text{otherwise} \end{cases}$$

Weight update: For all $i' (\neq i)$ such that $O_{i'j} \neq *$

$$w_{ii'} := \begin{cases} (2-\gamma)w_{ii'} & \text{if } O_{i'j} \in V(\hat{M}_{ij}) \\ \gamma w_{ii'} & \text{if } O_{i'j} \notin V(\hat{M}_{ij}) \end{cases}$$

Note in the above (and else-where) that C_0 is the default value which is used to predict when no relevant observations have been made. In all the filtering methods we describe here and in all of our experiments, we set $C_0 = 3$. Also in our experiments we set $V(a) = \{x \in A : a-1 \leq x \leq a+1\}$.

We also consider an extension of G-Learn-Relation, which we call Cross-G-Learn-Relation, which makes use of observed values in the same row and different columns, in addition to those in the same column and different rows. (This is a generalization of the two-dimensional weighted majority algorithm called WMP2 proposed in [ALN95].)

Cross-G-Learn-Relation ($0 \leq \gamma < 1$)

With each row pair (i, i') is associated a weight, $w_{ii'}$, and with each column pair (j, j') is associated a weight, $w_{jj'}^c$.

Initialization: $w_{ii'} = w_{jj'}^c := 1$

Prediction:

$$\hat{M}_{ij} = \begin{cases} \arg \max_{a \in A} \left(\sum_{i': O_{i'j} \in V(a)} w_{ii'} + \sum_{j': O_{ij'} \in V(a)} w_{jj'}^c \right) & \text{if } \{i' : O_{i'j} \neq *\} \cup \{j' : O_{ij'} \neq *\} \neq \emptyset \\ C_0 (\text{a constant}) & \text{otherwise} \end{cases}$$

Weight update: Update $w_{ii'}$ as in G-Learn-Relation and additionally for all $j' (\neq j)$ such that $O_{ij'} \neq *$,

$$w_{jj'}^c := \begin{cases} (2-\gamma)w_{jj'}^c & \text{if } O_{ij'} \in V(\hat{M}_{ij}) \\ \gamma w_{jj'}^c & \text{if } O_{ij'} \notin V(\hat{M}_{ij}) \end{cases}$$

Next the version of G-Learn-Relation in which we incorporate indirect evidence, referred to as Learn-Relation-IE, enhances the weights used in G-Learn-Relation by taking into account indirect evidence. If we let $d_{ii'} = C_{ii'} - W_{ii'}$ with $C_{ii'}$ and $W_{ii'}$ as defined above, then roughly speaking $d_{ii'} > 0$ is evidence for row i being similar to row i' , and $d_{ii'} < 0$ for the converse. If, for some third row i'' , we have both $d_{ii''} > 0$ and $d_{i'i''} > 0$, then this can be used as indirect evidence for i and i' being similar. Conversely, if we have $d_{ii''} \cdot d_{i'i''} < 0$, then this is indirect evidence for i and i' being dissimilar. Thus, we redefine the weights of G-Learn-Relation by adding $\delta \cdot \min\{|d_{ii''}|, |d_{i'i''}|\}$ to $C_{ii'}$ if $d_{ii''} > 0$ and $d_{i'i''} > 0$, and adding $\delta \cdot \min\{|d_{ii''}|, |d_{i'i''}|\}$ to $W_{ii'}$ if $d_{ii''} \cdot d_{i'i''} < 0$, where δ is a small constant controlling the degree of contribution of indirect evidence. The rest of the algorithm (prediction and direct weight update) is the same as G-Learn-Relation.

Learn-Relation-IE ($0 \leq \gamma < 1, 0 \leq \delta$)

With each row pair (i, i') is associated counters $C_{ii'}$, $W_{ii'}$.

Initialization: $C_{ii'} = W_{ii'} := 0$

Prediction: Predict as in G-Learn-Relation, except the weights $w_{ii'}$ are calculated as follows.

$$\begin{aligned} d_{ii'} &= C_{ii'} - W_{ii'} \\ e_{ii'} &= C_{ii'} + \delta \sum_{d_{ii''} > 0, d_{i'i''} > 0} \min\{|d_{ii''}|, |d_{i'i''}|\} \\ f_{ii'} &= W_{ii'} + \delta \sum_{d_{ii''} \cdot d_{i'i''} < 0} \min\{|d_{ii''}|, |d_{i'i''}|\} \\ w_{ii'} &= (2-\gamma)^{e_{ii'}} \gamma^{f_{ii'}} \end{aligned}$$

Update: For all $i' (\neq i)$,

$$\begin{aligned} C_{ii'} &:= C_{ii'} + 1 & \text{if } O_{i'j} \in V(\hat{M}_{ij}) \\ W_{ii'} &:= W_{ii'} + 1 & \text{if } O_{i'j} \notin V(\hat{M}_{ij}) \end{aligned}$$

We compare the performance of these generalized weighted majority algorithms against standard methods based on correlation coefficients. Here we informally describe these methods and refer the interested reader to [SM95] for detailed definitions. Like G-Learn-Relation, the correlation-based methods make predictions by weighted voting by different rows for

which the entries in the same column have been observed, except the weights between the rows are computed using 'correlation coefficients.' The correlation coefficient between any pair of rows is calculated using the observed values in those rows in common columns. Following [SM95], we also consider three variants of the basic correlation-based method (also known as **Pearson r** method): The thresholded method (**Pearson r** $L = \theta$) which lets only rows with a correlation coefficient higher than threshold θ vote; the constrained method (**constrained Pearson r**) which fixes the average in calculating the correlation coefficients at a constant rather than calculating it from the data (in our experiments it was fixed at 3), and the combination of the two; the thresholded constrained method (**constrained Pearson r** $L = \theta$).

4 Experiments

4.1 The data

In our experiments, we made use of artificially constructed (simulated) data, as well as real data obtained through actual experiments on collaborative filtering in a patent clipping service [AI97]. The simulation data we used were for a target matrix of size 100 by 100 with 5 row types and 5 column types, with noise added. We first generated a 5 by 5 matrix M_b by randomly assigning one of four groups $\{1, 2\}$, $\{2, 3\}$, $\{3, 4\}$, $\{4, 5\}$ to each of its entries. Then, based on M_b , we generate a 100 by 100 matrix M by randomly assigning one of the five rows of M_b to each row of M , and one of the five columns of M_b to each column of M . Finally, we introduce noise by probabilistically assigning one of five scores (1 through 5) to each group according to the following probability table. For example, if row i' of M_b is assigned to row i of M , column j' of M_b is assigned to column j of M , and the group assigned to the i', j' -entry of M_b is $\{2, 3\}$, then, the scores 1, 2, 3, 4 and 5 are assigned to the i, j -entry of M with probability $\frac{\alpha}{2}$, $\frac{1}{2} - \frac{\alpha}{2}$, $\frac{1}{2} - \frac{\alpha}{2}$, $\frac{\alpha}{2}$ and $\frac{\alpha^2}{2}$, respectively.

entry value of M_b	probability of assigning each score				
	1	2	3	4	5
$\{1, 2\}$	$\frac{1}{2}$	$\frac{1}{2} - \frac{\alpha}{2}$	$\frac{\alpha}{2} - \frac{\alpha^2}{2}$	$\frac{\alpha^2}{2} - \frac{\alpha^3}{2}$	$\frac{\alpha^3}{2}$
$\{2, 3\}$	$\frac{\alpha}{2}$	$\frac{1}{2} - \frac{\alpha}{2}$	$\frac{1}{2} - \frac{\alpha}{2}$	$\frac{\alpha}{2} - \frac{\alpha^2}{2}$	$\frac{\alpha^2}{2}$
$\{3, 4\}$	$\frac{\alpha^2}{2}$	$\frac{\alpha}{2} - \frac{\alpha^2}{2}$	$\frac{1}{2} - \frac{\alpha}{2}$	$\frac{1}{2} - \frac{\alpha}{2}$	$\frac{\alpha}{2}$
$\{4, 5\}$	$\frac{\alpha^3}{2}$	$\frac{\alpha^2}{2} - \frac{\alpha^3}{2}$	$\frac{\alpha}{2} - \frac{\alpha^2}{2}$	$\frac{1}{2} - \frac{\alpha}{2}$	$\frac{1}{2}$

In our experiments, we set $\alpha = 0.1$, which translates to noise rate of 0.075.

The real data we used in our experiments are scores given by various people on patents according to their interests. Scores were given by 77 people on 2558 patents, with about 5.4 per cent of the entries filled. Since on those entries for which few related entries are

known, none of the filtering methods considered here would do well, we extracted a portion of this data by restricting the people to those who scored at least 50 patents, and the patents to those that were rated by at least 10 people. This resulted in reducing the matrix to 48 people by 277 patents, and 29 per cent of this smaller matrix was filled. The scores, which are integers between 1 and 5, are distributed as follows.

1	2	3	4	5	Total
1541	659	590	600	425	3815

Note that the higher the score of a patent is the more interesting it is perceived by the user who scored it.

In our experiments involving simulated data, the performance of each method was averaged over 4 randomized runs on 4 randomly generated target matrices, 16 runs in total. For the experiments with real data, we took average over 4 randomized runs for each method.

4.2 Comparison with Correlation-based methods

First, we compared the predictive performance of G-Learn-Relation against those of the four variants of correlation-based methods described in Section 3. For the threshold value in a thresholded method, and for the value of γ in G-Learn-Relation(γ), we tried all multiples of 0.1 between 0 and 1. The results are shown in Figure 1. Among the correlation-based methods, it is verified that the thresholded constrained Pearson r method did the best, as reported in [SM95]. It is clear that G-Learn-Relation out-performed all of these methods on real data, and it was essentially tied with the best of all the correlation-based methods on simulated data. On real data, this tendency is more evident for column(patent)-based methods, but as the column-based methods perform better than the row-based methods, G-Learn-Relation is clearly the best performing method overall.

We also evaluated these methods using measures that are more often used in practical applications, *precision* and *recall*. Figure 2 compares precision and recall (in the last 200 trials at each trial) for the two thresholded correlation-based methods and G-Learn-Relation. Figure 3 plots a combination of these two measures called 'F-measure' (more precisely $F_{\beta=1}$ in [Lewis94]), namely $\frac{2PR}{P+R}$, where P stands for precision and R for recall.

These graphs were obtained using real data using similarity between patents. We considered the entries that were given the score of 5 as *desirable* and predicted valued of at least 3.5 to be *selected*.³ We can see that G-Learn-Relation achieves the highest recall rate and

³The precision is calculated as $\frac{N_s}{N_t}$, where N_s is the number of selected entries and N_t is the number of *correctly*

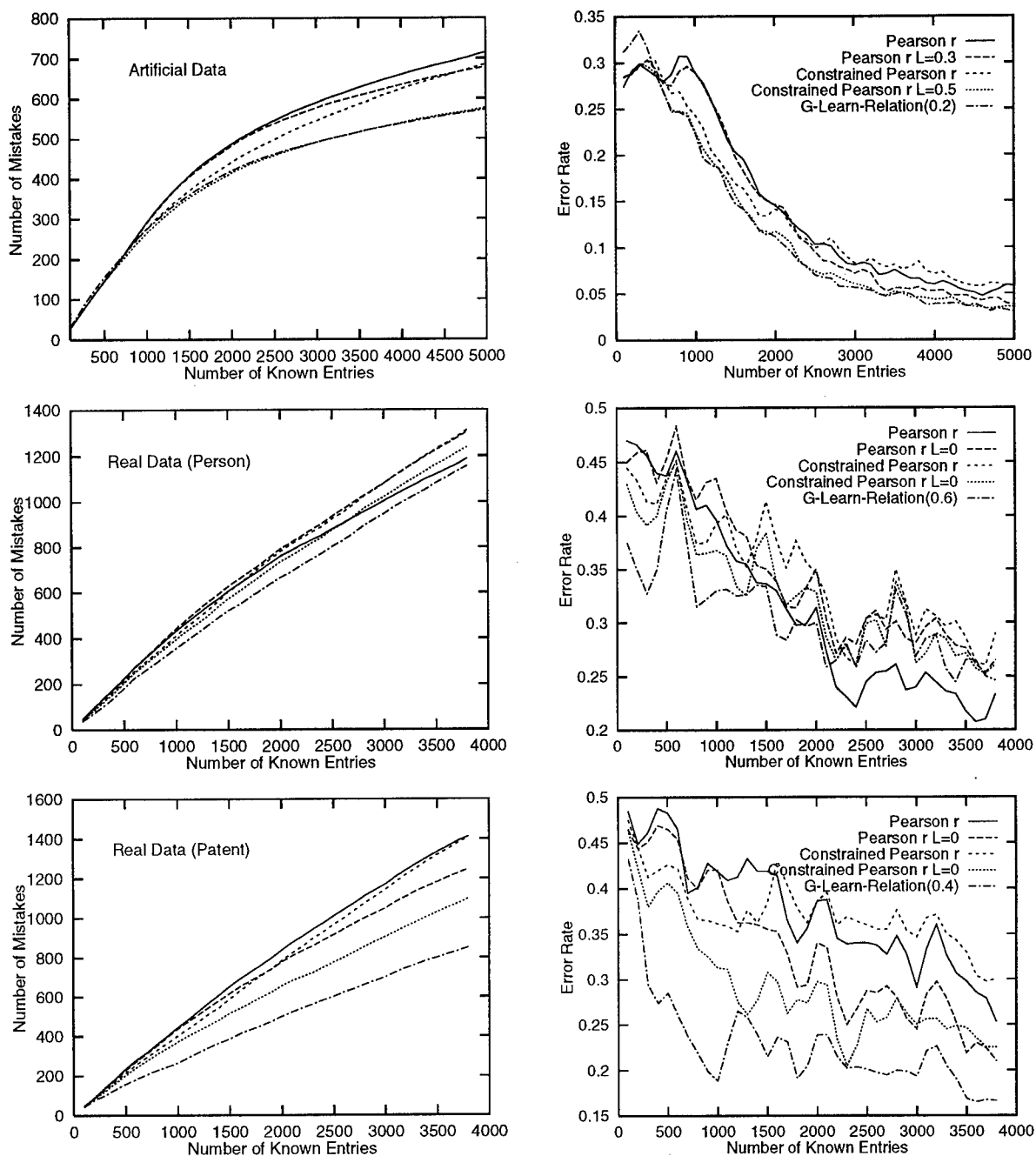


Figure 1: Correlation-based methods vs. G-Learn-Relation: Left graph: cumulative number of mistakes; Right graph: error rate in the last 200 trials. Top: simulated data; Middle: Real data (similarity between people); Bottom: Real data (similarity between patents).

the precision is comparable to others. Note that the thresholded constrained Pearson method which enjoys

selected entries of those. The recall is $\frac{N_c}{N_d}$, where N_d is the number of *desirable* entries.

the highest precision suffers from having a very low recall rate.

Another desirable aspect of G-Learn-Relation is its relative insensitivity towards the exact choice of its pa-

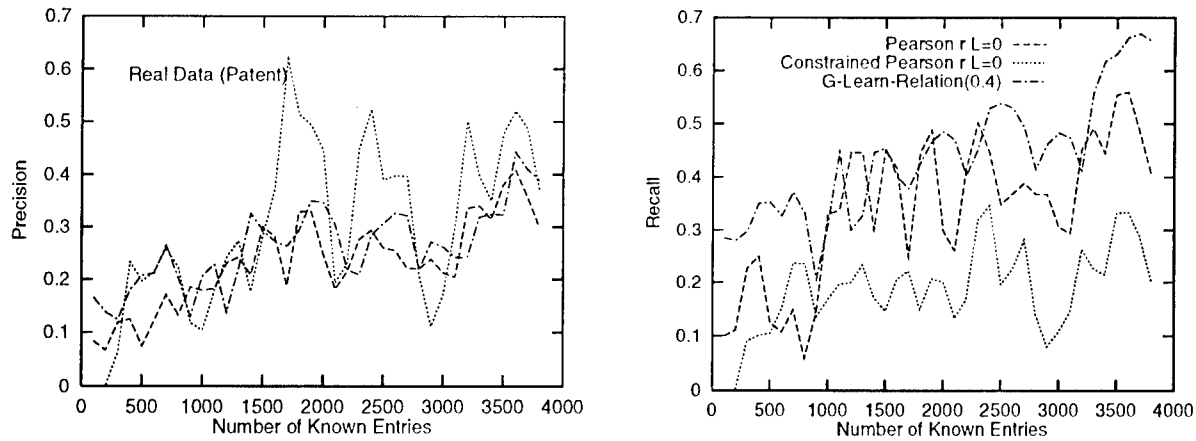


Figure 2: Precision(left) and recall(right) in the last 200 trials.

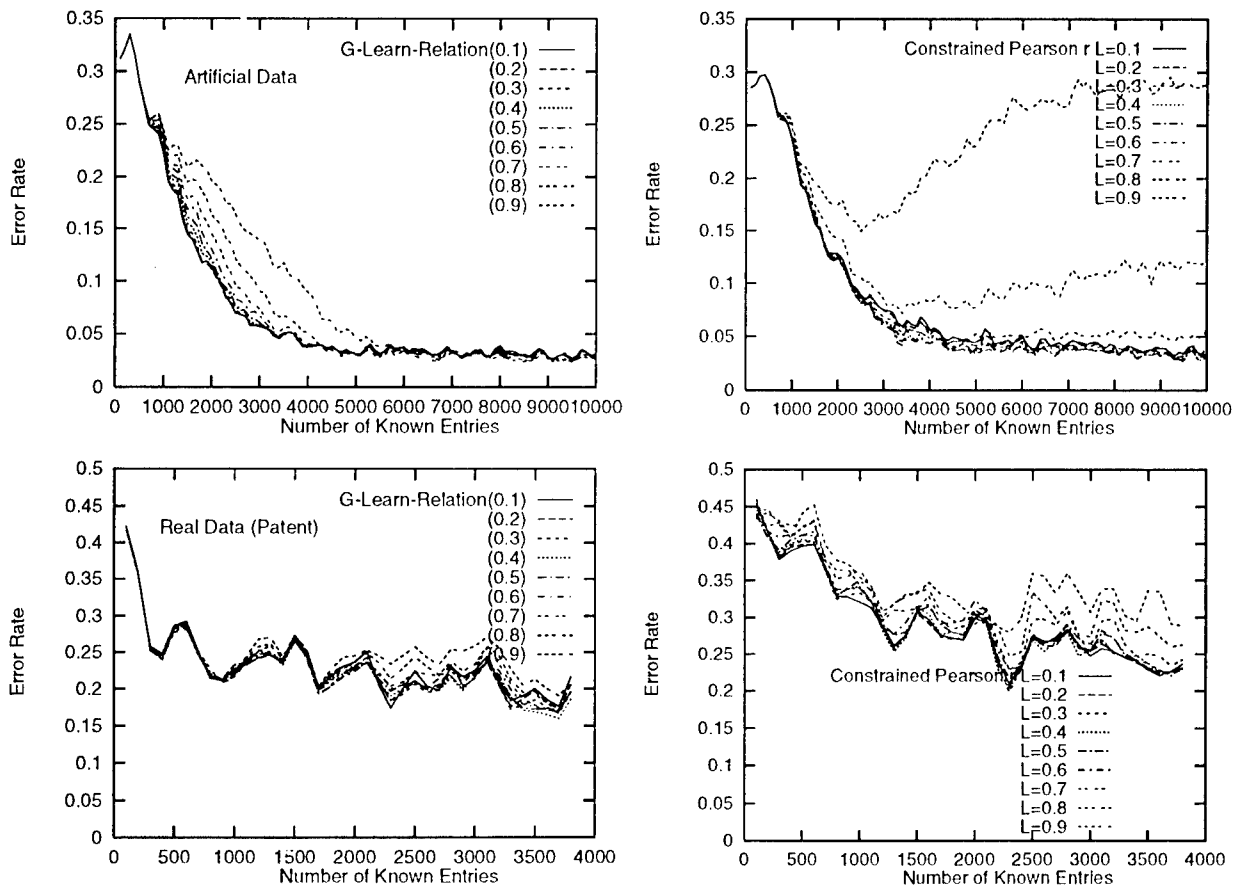


Figure 4: Performance dependence on the choice of parameters: Top: simulated data; Bottom: Real data (similarity between patents). Left: G-Learn-Relation; Right: Thresholded constrained correlation method.

parameter (γ .) Figure 4 shows how the predictive performance of G-Learn-Relation and the thresholded constrained correlation method vary, as we change their parameters. From the data we can see that the perfor-

mance of the thresholded correlation-based method is extremely sensitive to small changes in the threshold value in a certain range. In contrast, small changes in γ of G-Learn-Relation do not significantly affect its pre-

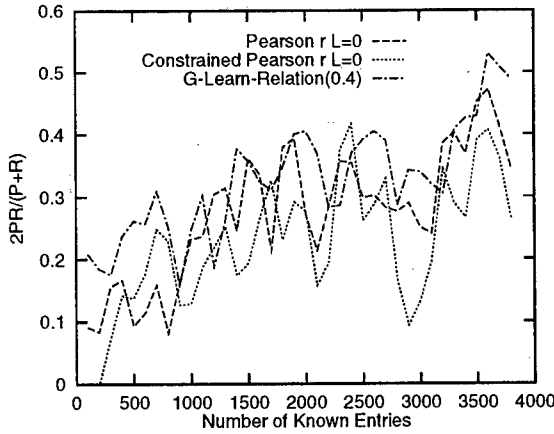


Figure 3: F-measure ($\frac{2PR}{P+R}$) in the last 200 trials.

dictive performance. This phenomenon is especially noticeable on the simulated data, although the same tendency is observed in the real data. In a practical application, a wrong choice of threshold could be costly for correlation-based methods.

4.3 Using both rows and columns

We evaluated the performance improvement brought about by the ‘cross-methods,’ namely methods that make use of both row similarity and column similarity, on G-Learn-Relation and the best performing correlation-based method – the thresholded constrained Pearson r method.

The results of this experimentation are shown in Figure 5. On the simulated data, it is observed that G-Learn-Relation is more radically by the cross method, and as a result its cross version clearly out-performs that of the Thresholded Constrained Pearson r. On the real data, however, the performance of the cross method (for both G-Learn-Relation and Constrained Pearson r) was comparable to that of the column-based method, although it was significantly better than the row-based method. This may be partly attributable to the asymmetry of the real data we used: there were only 48 rows whereas there were 277 columns. In practical applications with more even numbers of rows and columns, the effect may be more visible.

4.4 Using indirect evidence

We compared the performance of Learn-Relation-IE and that of G-Learn-Relation, as well as their respective ‘cross’ versions. Of the two parameters γ, δ in Learn-Relation-IE(γ, δ), the same choice of γ was used as G-Learn-Relation, and the best choice (out of a few) was used for δ . The results are shown in Figure 6. On the simulated data, it is observed that the performance is improved significantly for both G-Learn-Relation and Cross-G-Learn-Relation, for a cer-

tain range of trial numbers; trials around 1,000th to 3,000th out of 10,000. On real data, unfortunately, no significant improvement was observed, except a little for the row(people)-based method. It may be that, with this particular data set, the range of trial numbers on which significant improvement is achieved is yet to come.

5 Theoretical analysis

In this section, we theoretically analyze the performance of G-Learn-Relation and that of correlation-based methods. In particular, we prove an upper bound on the worst case number of mistakes made by G-Learn-Relation. We also show a lower bound on the worst case number of mistakes made by the correlation-based method, which shows that, as a learning method, the correlation-based method does not necessarily converge, and can make a huge number of mistakes in the worst case.

5.1 Mistake bound for G-Learn-Relation

It can be shown that the upper bound obtained by Goldman and Warmuth for Learn-Relation can be generalized for G-Learn-Relation, when the target function is real-valued and $V(a)$ is defined as $V(a) = V_d(a) \triangleq \{x : a - d \leq x \leq a + d\}$.

We need a few definitions to state our result. Let p be a partition over the set of rows R , k_p the size of the partition, and $p = \{S^1, \dots, S^{k_p}\}$. Then let $n_i = |S^i|$ and $S_j^i = \{M_{rj} : r \in S^i\}$. Let $\mathcal{N}_a(S_j^i)$ be the number of $r \in S^i$ such that $M_{rj} \in V_d(a)$, and define

$$\delta_{ij} = n_i - \max_a \mathcal{N}_a(S_j^i).$$

Let the set of columns be $\{1, \dots, m\}$. Let $\delta_i = \sum_{j=1}^m \delta_{ij}$ and define the noise α_p of partition p as $\alpha_p = \sum_{i=1}^{k_p} \delta_i$. Then, the following theorem holds.

Theorem 1 For all $\gamma \in [0, 1]$, Algorithm G-Learn-Relation(γ) makes at most

$$\min_p \left\{ k_p m + \min \left\{ \frac{\frac{n^2}{2e} \log e + \alpha_p (n - \frac{\alpha_p}{k_p m}) \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}, \sqrt{\frac{3mn^2 \log k_p + 2\alpha_p (mn - \alpha_p) \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}} \right\} \right\}$$

mistakes. Here, $\beta = \frac{\gamma}{2-\gamma}$, and the first minimization is with respect to all possible partitions p satisfying the following condition:

$$\delta_i \leq n_i m / 2 \text{ for all } i = 1, \dots, k_p. \quad (1)$$

The proof is similar to the proof of the analogous theorem for Learn-Relation in [GW95], and omitted due

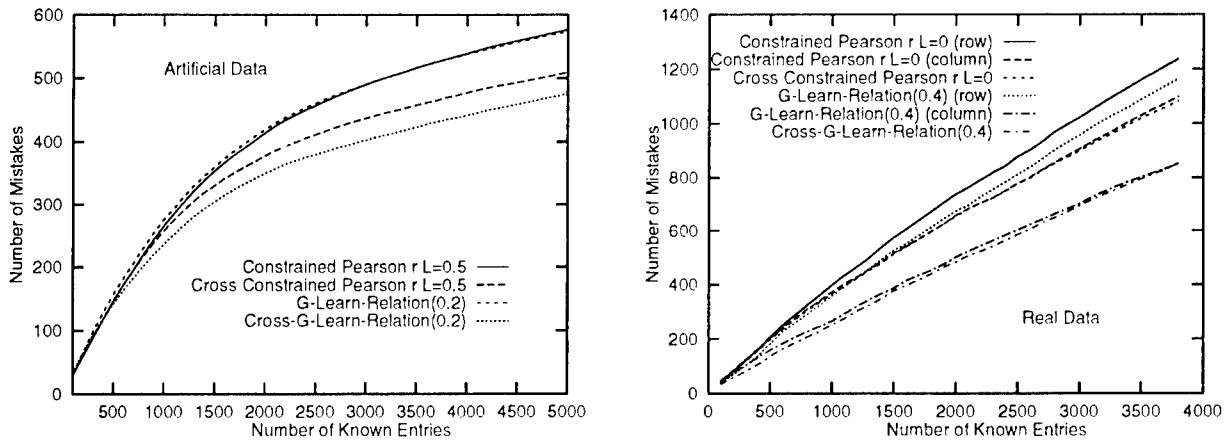


Figure 5: Effect of Cross methods: Left: simulated data; Right: real data

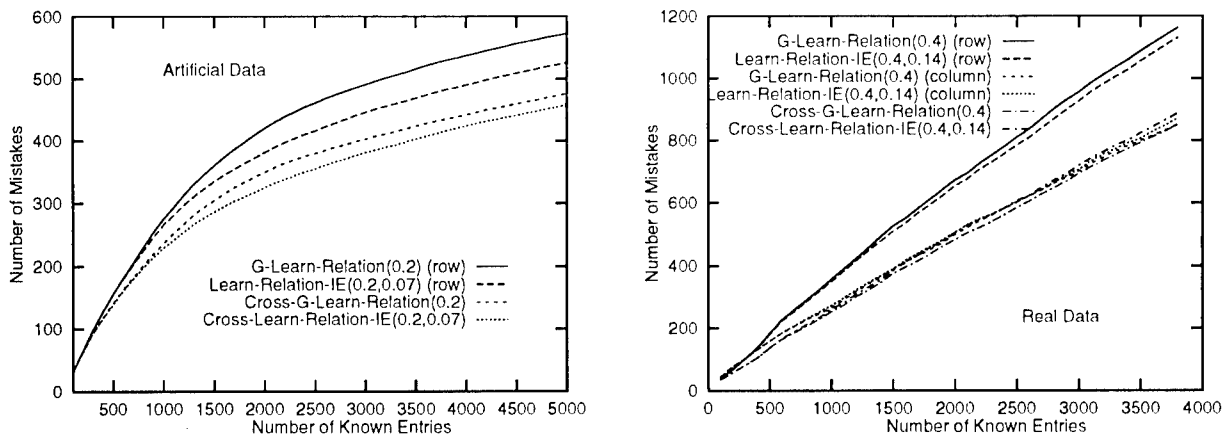


Figure 6: Performance of G-Learn-Relation and Learn-Relation-IE: Left: simulated data; Right: real data.

to lack of space. The condition (1) states that more than half the elements in each partition assumes a representative value. This is reasonable, and note in particular, that it always holds when the target function is binary. Now, by plugging in $\gamma = 0, \alpha_p = 0$ in the above theorem, we obtain the following as corollary.

Corollary 1 For any noise-less partition p (with $\alpha_p = 0$), Algorithm *G-Learn-Relation(0)* makes at most

$$k_p m + \min \left\{ \frac{n^2}{2\epsilon} \log \epsilon, \sqrt{3mn^2 \log k_p} \right\}$$

mistakes.

5.2 A mistake lower bound for the correlation-based methods

We show that, in the absence of noise, the correlation-based method can make a lot more mistakes in the worst case than *G-Learn-Relation(0)* with $V(a) = \{a\}$.

For this analysis, we assume that a prediction is correct only when the round-off value equals the correct value.

Theorem 2 In the worst case, any of the four correlation-based methods can make as many as nm/C' mistakes, where n is the number of rows and m is the number of columns, and C' is a positive constant.

Proof

We first prove the statement for the basic correlation-based method with no threshold and no constraint. Suppose that the target matrix consists of many repetitions (in both row and column directions) of the following block consisting of two types of rows and four types of columns, where i ranges over 0 to $m/8 - 1$.

Column No.	$8i+1$	$8i+2$	$8i+3$	$8i+4$	$8i+5$	$8i+6$	$8i+7$	$8i+8$
Type 1	5	1	5	1	5	1	5	1
Type 2	1	5	5	1	5	1	5	1

Thus if the target matrix were n by m , it would consist of $n/2$ rows of type 1 and $n/2$ rows of type 2, and each type of row would consist of $m/8$ repetitions of 8 columns. Suppose that the trials proceed from left to right and top down by blocks. Within each block, the trials proceed by column, *except* the first two columns ($8i + 1$ -st and $8i + 2$ -nd columns) are predicted in the row-first ordering. That is, after predicting the $8i + 1$ -st column in the type 1 row, the $8i + 2$ -nd column in the same row is predicted before the two columns in the other (type 2) row are predicted. Note, with this ordering, that when predicting an $8i + 1$ -st column of any block, the average of the past values for any rows is 3. It can be shown that, when the correlation-based method is predicting an entry in the $8i + 1$ -st column, the predicted value will not exceed $3 + 2/3$ for a row of type 1, and it will be at least $3 - 2/3$ for a type 2 row. This is because the number of known entries in a row of the same type does not exceed the number of known entries in a different type of row. Thus, the correlation-based method makes a mistake on every entry in the $8i + 1$ -th column. Hence, if n is the number of rows and m is the number of columns, it will make at least $nm/8$ mistakes.

Note that the above argument applies on the constrained correlation-based method, since the average is fixed at 3. A thresholded correlation-based method can beat the above example by setting the threshold to be higher than $1/2$, but for any fixed value of threshold, an analogous example can be constructed by making the block longer, repeating the last two columns an appropriate number of times. This would yield a similar bound, except a different constant replaces 8. \square

In contrast, we know from Corollary 1 that G-Learn-Relation(0) makes at most $2m + n\sqrt{3m}$ mistakes. Note that as m and n become large, the final error rate of G-Learn-Relation(0) will approach zero (the learning converges), but the error rate of the correlation-based method (in this worst case) will not be lower than $1/C$.

6 Concluding remarks

We have applied weighted majority type prediction algorithms on the problem of collaborative filtering, and empirically demonstrated that they perform better than the correlation-based filtering methods. In so doing, we proposed a generalization G-Learn-Relation of the weighted majority prediction algorithm of Goldman and Warmuth [GW95] to the multi-valued setting, and gave a theoretical performance guarantee on the performance of this algorithm. Additionally, we proposed a version of G-Learn-Relation that makes use of indirect evidence, as well as a version in which both row similarity and column similarity are used for prediction. In both cases, significant performance improvement was observed in experiments involving sim-

ulated data. It is left as future research to verify the same on real data, which we believe will require larger-scale experiments.

Acknowledgement

We thank Dr. S. Goto and Dr. S. Doi of C & C Media Research Laboratories, NEC for their support and encouragement. We also thank Dr. Y. Ariyoshi and Mr. T. Ichiyama of Human Media Research Laboratories, NEC for providing the patent clipping data.

References

- [AI97] Y. Ariyoshi and T. Ichiyama, An information filtering method combined content based filtering and social information filtering, IEICE 8th Workshop on data engineering, pp.49-54. 1997.
- [ALN95] N. Abe, H. Li and A. Nakamura. On-line Learning of Binary Lexical Relations Using Two-dimensional Weighted Majority Algorithms. *Proc. of 12th Int. Conf. on Machine Learning*, 1995, pp.3-11.
- [GRS93] S. Goldman and R. Rivest and R. Schapire. "Learning Binary Relations and Total Orders." *SIAM J. of Comput.* 22(5), 1993, pp.1006-1034.
- [GW95] S. Goldman and M. Warmuth. "Learning Binary Relation Using Weighted Majority Voting." *Machine Learning* 20, 1995, pp.245-271.
- [LB97] K. Lang and E. Baum. Better average-case predictions for k-binary relations. Unpublished manuscript, 1997.
- [Lewis94] D. D. Lewis and W. A. Gale. A Sequential Algorithm for Training Text Classifiers. *Proc. of 17th Annual International ACM-SIGIR Conf. of Res. and Dev. in Information Retrieval*, 1994, pp. 3-12.
- [NA95] A. Nakamura and N. Abe. On-line Learning of Binary and n -ary Relations over Multi-dimensional Clusters. *Proc. of COLT '95*, 1995, pp.214-221.
- [RISBR94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proc. of CSCW*, 1994, pp.175-186.
- [SM95] U. Shardanand and P. Maes. Social Information Filtering: Algorithms and Automating "Word of Mouth". *Proc. of CHI95*, 1995, pp.210-217.

On Feature Selection: Learning with Exponentially many Irrelevant Features as Training Examples

Andrew Y. Ng
 Artificial Intelligence Laboratory
 Massachusetts Institute of Technology
 Cambridge, MA 02139
 ayn@ai.mit.edu

Abstract

We consider feature selection in the “wrapper” model of feature selection. This typically involves an NP-hard optimization problem that is approximated by heuristic search for a “good” feature subset. First considering the idealization where this optimization is performed exactly, we give a rigorous bound for generalization error under feature selection. The search heuristics typically used are then immediately seen as trying to achieve the error given in our bounds, and succeeding to the extent that they succeed in solving the optimization. The bound suggests that, in the presence of many “irrelevant” features, the main source of error in wrapper model feature selection is from “overfitting” hold-out or cross-validation data. This motivates a new algorithm that, again under the idealization of performing search exactly, has sample complexity (and error) that grows *logarithmically* in the number of “irrelevant” features – which means it can tolerate having a number of “irrelevant” features *exponential* in the number of training examples – and search heuristics are again seen to be directly trying to reach this bound. Experimental results on a problem using simulated data show the new algorithm having much higher tolerance to irrelevant features than the standard wrapper model. Lastly, we also discuss ramifications that sample complexity logarithmic in the number of irrelevant features might have for feature design in actual applications of learning.

1 Introduction

In recent years, Feature Selection for classification and regression has been enjoying increasing interest in the Machine Learning community. Impressive performance gains have been reported by numerous authors, and numerous feature subset search heuristics have been proposed. (The literature is too wide to survey here, but see [Langley, 1994] and [Miller, 1990] for overviews.) In view of these significant empirical successes, one central question is: What theoretical justification is there for feature selection? For example, in parametric function approximation schemes such as linear regression, it is often the case that excluding a feature is mathematically identical to setting the coefficient(s) associated with that feature to 0. As feature selection typically runs a risk of misidentifying the “irrelevant” features, why then is it apparently often superior to try to estimate which features are “irrelevant” and set their coefficients to 0, rather than leave them and use the estimated coefficients for these features (which will typically be near 0 anyway)? The theoretical results in this paper will address this question.

Since feature selection attempts to eliminate “irrelevant” features, another central question is: How does the performance of feature selection scale with the number of irrelevant features? The Winnow algorithm of Littlestone for learning Boolean monomials, or more generally also k -DNF formulae and r -of- k threshold functions (over boolean inputs), from noiseless data enjoys worst-case loss logarithmic in the number of irrelevant features [Littlestone, 1988]. Likewise, the EG algorithm for linear regression with quadratic error also has such loss (and indeed sample complexity) that grows logarithmically in the number of irrelevant features [Kivinen and Warmuth, 1994]. For learning from noiseless data, of a representation of a boolean concept

(over boolean inputs), Almuallim and Dietterich have also shown that an algorithm that finds the smallest set of features consistent with the training data (such as by exhaustive enumeration) also enjoys loss logarithmic in the number of irrelevant features [Almuallim and Dietterich, 1994]. If it were true in general that feature selection makes sample complexity logarithmic in the number of irrelevant features (though possibly depending more heavily on the number of relevant features), then this would imply, for example, that *squaring* the number of features we have means needing only *twice* as much training data. This could have huge ramifications on the way features are designed for real-world applications. In this paper, we will show that, modulo computational and approximation issues, this ideal of *logarithmic* sample complexity in the number of irrelevant features – which of course means being able to handle *exponentially* many irrelevant features as training examples – can indeed be achieved with a new feature selection algorithm we propose.

Next, the notion of “relevance” is closely related to feature selection. Intuitively, one goal of feature selection is to eliminate all but a small set of “relevant” features, which are then given to an induction algorithm. However, there have been difficulties with a number of definitions of “relevance” [Kohavi and John, 1997], and we take the alternative view, which is quite similar in flavor to those in [Littlestone, 1988] and [Kivinen and Warmuth, 1994], of the goal of feature selection as this: If there exists a hypothesis that, using only a “small” number of features, gives good generalization error, then we want our classifier to achieve close to this level of performance with high probability. This will be made rigorous in subsequent sections, but note in particular that we make no claims towards excluding “irrelevant” features or including all the “relevant” features, so long as the particular set of selected features allows us to have performance close to that of using the “optimal” set of features.¹ In the remainder of this paper, we will use the terms “relevant” and “irrelevant” only when we expect them to be consistent with any reasonable definition of relevance.

Using the terminology introduced by [John et al., 1994], feature selection algorithms broadly fall into the “filter” and the “wrapper” models. The filter model relies on general characteristics of the training

data to select some feature subset, doing so without reference to the learning algorithm. In the wrapper model, one generates sets of candidate features, runs them through the learning algorithm, and uses the performance of the resulting hypothesis to evaluate the feature set. While the wrapper model tends to be more computationally expensive, it also unsurprisingly tends to find feature sets better suited to the inductive biases of our learning algorithm, and tends to give superior performance [Langley, 1994]. In this paper, we study only the wrapper model of feature selection, and largely in the context of classification.

Our analysis is largely inspired by [Kearns, 1996], with our theoretical results heavily based on the techniques given there and those outlined in [Kearns et al., 1997]. We also rely heavily on tools from [Vapnik, 1982], that give a very general framework for bounding the deviation of training error from generalization error.

2 Preliminaries.

2.1 Feature Selection

Let X be the fixed f -dimensional input space, where f is the number of features in the inputs we are provided. For simplicity, we also assume a fixed binary concept $c : X \mapsto \{0, 1\}$. We are provided m training examples $S = \{x^i, y^i\}_{i=1}^m$, with each of the f -dimensional input vectors $x^i = [x_1^i \ x_2^i \ \dots \ x_f^i]^T$ drawn *i.i.d.* from some fixed distribution D_X over X , and corresponding labels $y^i = c(x^i) \in \{0, 1\}$. In this development, we will also briefly consider the case where the labels are independently corrupted by noise with a noise rate $\eta \in [0, 0.5)$, so that $y^i = c(x^i)$ with probability $1 - \eta$, and $y^i = 1 - c(x^i)$ with probability η . Note that c may use all f features, but we hope that it can be approximated well (in the generalization-error sense, to be defined shortly) by a function that depends only on a small subset of the f features.

We will use uppercase F to denote sets of features, and use F_i to identify the i -th feature. For example, the feature set including the 1st, 4th and 10th features may be written $F = \{F_1, F_4, F_{10}\}$. For any input vector x , let $x|_F$ be x with all the features not in F eliminated; sometimes, we will call this “ x restricted to F .” Analogously, let $X|_F$ denote the input space X with all the dimensions/features not in F eliminated, and $S|_F$ be the data set S with each x^i replaced by $x^i|_F$. In a slight abuse of notation, if we have a hypothesis $h : X|_F \mapsto \{0, 1\}$ defined only the subspace of features $X|_F$, we extend it to X in

¹Aside from good generalization error, other goals of feature selection might be user-interpretability and parsimony of hypotheses for fast prediction. We will not address these goals in this paper.

the natural way (with h ignoring features not in F). Thus, for any hypothesis h , we can write the generalization error (with respect to uncorrupted data) as $\varepsilon(h) = \Pr_{x \in D_X}[h(x) \neq c(x)]$ (where the dependence of $\varepsilon(h)$ on D_X has been suppressed for notational brevity,) and the empirical error on a set of data S as $\hat{\varepsilon}_S(h) = \frac{1}{|S|} |\{(x, y) \in S | h(x) \neq y\}|$.

2.2 The wrapper model

In the wrapper model of feature selection suggested by [John et al., 1994], we are given a learning algorithm L that, for any set of features F , takes a training set $S|_F$, and outputs a hypothesis $h : X|_F \rightarrow \{0, 1\}$. Given a training set S , an application of feature selection under this model might randomly split S into a training set S' of size $(1 - \gamma)m$ and a hold-out set S'' of size γm , and perform a search for a set of features F so that when the learning algorithm is applied to S' restricted to F , the resulting hypothesis $h = L(S'|_F)$ has low empirical error $\hat{\varepsilon}_{S''}(h)$ on the hold-out data S'' . Here, $\gamma \in [0, 1]$, the fraction of S assigned to the hold-out set, is called the hold-out fraction. A more sophisticated application of feature selection may use n -fold or leave-one-out cross validation rather than hold-out. But as they asymptotically yield at best small-constant improvements over using hold-out and as leave-one-out is at worst little better than training error in estimating generalization error, while rendering the algorithm's performance much less tractable to analysis [Kearns and Ron, 1997], we will not explicitly consider them here, though we believe our results will be suggestive of the performance of these schemes as well.

For any given learning algorithm L , the optimal way to perform feature selection is intimately related to the inductive biases of L . For example, if L is "sufficiently clever" about doing its own feature selection, then one would simply give it S unrestricted to any feature subset, and allow it to select its own features. For this analysis, therefore, we make the (rather strong) assumption that given a particular data set $S|_F$, L chooses the hypothesis h from some class of hypotheses (shortly to be formalized) so as to minimize training error. This closely ties in with the learning framework studied by [Vapnik, 1982], and is also used in [Kearns, 1996] and [Kearns et al., 1997] in proving bounds on generalization error. We believe it to be a very natural model, and that it is a rich enough class of learning algorithms to merit detailed study. (But also see [Kearns et al., 1997] for comments regarding relations to learning algorithms that do not exactly do this; for example,

it is not difficult to derive rigorous generalizations of all of our results if L manages to only approximately minimize training error.)

More formally, for any feature set F , we assume that we have a hypothesis class H_F , of hypotheses each with domain $X|_F$. But, with many induction algorithms, each feature is treated in a "similar" manner – for example, when $X = \mathcal{R}^J$, then for two feature sets F and F' of the same size, it makes intuitive sense to identify $X|_F$ and $X|_{F'}$ and therefore H_F and $H_{F'}$, as they are both sets of functions mapping from $\mathcal{R}^{|F|}$ to $\{0, 1\}$. For simplicity, let us further make the assumption that the hypothesis class H_F depends on F only through $|F|$, and let H_r be our set of functions with domain X restricted to any set of r features. (This assumption is not really necessary, but it greatly eases our notational burden, and leaving out the assumption does not gain much in terms of theoretical results.) It will always be clear from context which particular set F of features $h \in H_{|F|}$ takes as input. Note also that we have assumed that there is some "uniform" way of handling all features, whether they are discrete/continuous, have different ranges, etc.. For simplicity, one may wish to think of the particular case where all features are real numbers for the remainder of this paper. In this notation then, our previous assumption of error minimization is that when L is given $S|_F$, it outputs the hypothesis $h \in H_F$ (where H_F is identified with $H_{|F|}$) that minimizes training error on $S|_F$. For the remainder of this paper, we will implicitly assume L meets these two assumptions – that it treats features "uniformly," and that it minimizes training error over $H_{|F|}$.

One more definition we need is to let r_{VC} be the Vapnik-Chervonenkis dimension [Vapnik and Chervonenkis, 1971, Vapnik, 1982] of the hypothesis class H_r . Normally, we expect $0_{VC} < 1_{VC} < 2_{VC} < \dots$, though this is not an assumption we use. For example, if H_r is the class of linear discriminant functions over \mathcal{R}^r , then $r_{VC} = r + 1$. We chose this notation so that, to specialize our ensuing bounds on generalization error to linear discriminant functions, which we later use in our experiments, r_{VC} may everywhere be replaced with r (or at least when $r > 0$).

Finally, to obtain the performance bounds, we wish to make statements of the form that "we will, with high probability, find a hypothesis with generalization error no worse than z more than the best hypothesis that uses r features." To formalize this, define the approximation rate function $\varepsilon_g(r)$ to be the *least* generalization error achievable by any hypothesis $h \in H_r$ using any set of r features. In general, we expect

$\varepsilon_g(1) \geq \varepsilon_g(2) \geq \dots$, though this is also not an assumption we require (except briefly when we summarize our results in terms of sample complexity).

Thus, in the common instantiation of wrapper model feature selection, we search for a feature set F such that when L is applied to $S'|_F$, the resulting hypothesis has low empirical error on the hold-out set. (That is, $\hat{\varepsilon}_{S''}(L(S'|_F))$ is minimized.) Leaving aside details of the actual search, we will call this idealization the STANDARD-WRAP algorithm. Note that in performing the search, enumeration over all the 2^f possible feature sets is usually intractable, and there is no known algorithm for otherwise performing this optimization tractably. Indeed, the Feature Selection problem in general is NP-hard [Garey and Johnson, 1979], but much work over recent years has developed a large number of heuristics for performing this search efficiently. (Again, the literature is too wide to survey here, but examples include [Moore and Lee, 1994, Caruana and Frietag, 1994, Yang and Hoavar, 1997], and [Langley, 1994, Miller, 1990] include overviews.) In this development, we will, in the style of [Kearns, 1996], give bounds for generalization error when this optimization is performed exactly. Of course, the extent to which our bounds predict actual performance will in part depend on the extent to which the optimization algorithms succeed in performing this search on "real life" distributions of data. Alternatively, one can also view these bounds as what the heuristic search/approximation algorithms are (in a rigorous sense, to be discussed later) aspiring to do, with the bounds giving insight into how we might expect the algorithms to perform.

3 Main Results

The ensuing bounds are all given to hold "with high probability." We defer their more detailed versions to the full paper, but note that when we say "with high probability," we mean that the bound holds with at least probability $1 - \delta$ for any $\delta > 0$, with constants that depend on δ (through an omitted $\log \frac{1}{\delta}$ term) hidden by the $O(\cdot)$ notation.

Bound for performance without feature selection

The Universal Estimate Rate bound of Vapnik and Chervonenkis [Vapnik and Chervonenkis, 1971, Vapnik, 1982] gives a bound on generalization error when learning using all f features without feature selection.

Theorem 1 (Vapnik and Chervonenkis, 1971)

With high probability, the generalization error of the hypothesis $\hat{h} = L(S)$, given by L applied to S (unrestricted to any feature subset), is bounded by:

$$\varepsilon(\hat{h}) \leq \varepsilon_g(f) + O\left(\sqrt{\frac{f_{VC}}{m}} \left(\log \frac{m}{f_{VC}} + 1\right)\right) \quad (1)$$

Note this is a bound for learning from noiseless data; when the training data labels have independently been corrupted at some noise rate η , the second term in the bound becomes $O\left(\sqrt{\frac{f_{VC}}{(1-2\eta)^2 m}} \left(\log \frac{m}{f_{VC}} + 1\right)\right)$.

Bound for performance of wrapper model

Applying the proof technique given in [Kearns, 1996] (used to bound the error of hold-out) to feature selection, we obtain the following theorem:

Theorem 2 *Given L, S, γ , the hypothesis \hat{h} output by STANDARD-WRAP, given by $\hat{h} = L(S'|_{\hat{F}})$ where $\hat{F} = \operatorname{argmin}_F \hat{\varepsilon}_{S''}(L(S'|_F))$, will, with high probability, have generalization error bounded by*

$$\varepsilon(\hat{h}) \leq \min_{0 \leq r \leq f} \left\{ \varepsilon_g(r) + O\left(\sqrt{\frac{r_{VC}}{(1-\gamma)m}} \left(\log \frac{m}{r_{VC}} + 1\right)\right) \right\} + O\left(\sqrt{\frac{f}{\gamma m}}\right) \quad (2)$$

Proof (Sketch): The first square-root term is simply the universal estimation rate bound as before, that says that with high probability, the hypothesis obtained by applying L to $S'|_F$ for any fixed F with $|F| = r$ will give additional error no more than $O\left(\sqrt{\frac{r_{VC}}{(1-\gamma)m}} \left(\log \frac{m}{r_{VC}} + 1\right)\right)$. Following this, using a holdout-test set of size γm to test 2^f hypotheses will, by a standard Chernoff-bound argument, result with high probability in picking a hypothesis with generalization error no more than $O(\sqrt{\log(2^f)/\gamma m}) = O(\sqrt{f/\gamma m})$ higher. \square

Again, this bound holds only when learning from noiseless data. Similar to Theorem 1, a generalization to learning from noisy data can be obtained by replacing all occurrences of m in any denominator term in the bound by $(1 - 2\eta)^2 m$, where η is the noise rate.

One important remark here is that the $O(\sqrt{f/\gamma m})$ term is a worst-case bound for evaluating 2^f hypotheses on the independent hold-out set S'' of size γm . Its increase with f reflects the fact that we are testing a set of hypotheses of size exponential in f , and that there is potential for “overfitting” the γm hold-out samples. (In the context of feature selection, the issue of overfitting of hold-out data was also raised by [Kohavi and Sommerfield, 1995]; see also [Ng, 1997] for a detailed discussion of overfitting of hold-out data in hypothesis selection.) But since this is a worst-case bound, it holds in particular for the “bad case” where all 2^f hypotheses are “very different” from each other. This is unlikely as they were trained on the same dataset S' and using only f distinct features. For at least some pathological hypothesis classes (that may, for example, include a set of hash-like basis function so that changing one feature’s range dramatically changes the output hypotheses,) this is certainly possible; but for more “sensible” hypothesis classes, we might expect it to be possible to significantly tighten this bound. We have not managed to formalize this yet, but conjecture, based on the behavior of power-law decay learning curves, that the asymptotic behavior for “many” learning algorithms will be better modeled by replacing this last term in the bound by $\sqrt{f^\alpha/\gamma m}$ for some $\alpha \in (0, 1]$. (A preliminary analysis suggests that under a (perhaps surprisingly large) range of formal modeling assumptions regarding how much hypotheses change when F is changed, the number of “significantly different” hypotheses does grow as $2^{O(f)}$, which would suggest $\alpha = 1$ behavior. On the other hand, there are certainly also some reasonable assumptions that would lead to $\alpha < 1$; and we defer a detailed discussion of this to the full paper.)

Bound for performance of new algorithm

For STANDARD-WRAP, the dependence on f of our bound on the error is $\sqrt{f/\gamma m}$ (or possibly $\sqrt{f^\alpha/\gamma m}$), and it comes from testing 2^f hypothesis on holdout-data. If $f \gg r_{VC}$ where r is the number of features needed to approximate the target concept well, this $\sqrt{f/\gamma m}$ will be the dominant term. Consider instead the following algorithm, which we call ORDERED-FS:

1. For each $0 \leq r \leq f$, find the hypothesis \hat{h}_r that, of all the hypotheses using exactly r features, minimizes error on the training set S' . (This involves a search over all sets of r features.)
2. Evaluate all $f+1$ hypotheses $\{\hat{h}_r\}_{r=0}^f$ on the hold-out set S'' , and pick the one with the smallest

hold-out error.

Note that we are now testing only $O(f)$ hypotheses on the hold-out data, so the previous $\sqrt{f/\gamma m}$ term now becomes $\sqrt{(\log f)/\gamma m}$.

Theorem 3 *Given L, S, γ , the hypothesis \hat{h} output by ORDERED-FS will, with high probability, have generalization error bounded by*

$$\epsilon(\hat{h}) \leq \min_{0 \leq r \leq f} \left\{ \epsilon_g(r) + O \left(\sqrt{\frac{r_{VC}}{(1-\gamma)m}} \left(\log \frac{m}{r_{VC}} + 1 \right) \right) + O \left(\sqrt{\frac{r \log f}{(1-\gamma)m}} \right) \right\} + O \left(\sqrt{\frac{\log f}{\gamma m}} \right) \quad (3)$$

Proof (Sketch): The first square-root term is simply the universal estimation rate bound as before, used to bound the additional error when training on any fixed feature set. For this to hold with probability $1 - \delta$, there is also an additive $(1/m) \log(1/\delta)$ within the square-root. Now, for any fixed r , we want to uniformly bound the deviation of training error from generalization error for all $\binom{f}{r}$ hypotheses that use exactly r features. Taking a standard union bound (see [Vapnik, 1982]), we replace $(1/m) \log(1/\delta)$ with $(1/m) \log(\binom{f}{r}/\delta)$, which (noting $\log \binom{f}{r} \leq r \log f$) gives the second term. Lastly, the third term comes, using a standard Chernoff-bound argument as before, from testing $O(f)$ hypotheses on the hold-out set of size γm . \square

Notice that, similar to STANDARD-WRAP, we have not explicitly addressed the NP-hard search problem for the optimal (here in the minimum training error sense) set of r features, and actual implementations of ORDERED-FS will generally have to rely on heuristic search. But for now, let us beg this computational issue and treat it similarly to how we had treated STANDARD-WRAP, appealing to the same approximations/idealizations as before, and also mentioning that, in a rigorous sense to be discussed later, the extent to which an approximation algorithm can solve the optimization is exactly the extent to which its error bound will reach the bound we give here, which means that our bound can as before be interpreted, in a formal sense, as being exactly what a heuristic search implementation is trying to attain. (In considering heuristic search implementations, it is also worth mentioning that searching to minimize training error is prob-

ably often somewhat easier than searching to minimize hold-out error, which STANDARD-WRAP requires; for example, in linear regression, we have fast algorithms for simultaneously evaluating training error for all single-feature changes to a feature subset.) This bound is also easily generalized to learning from noisy examples (again by replacing all occurrences of m in any denominator term with $(1 - 2\eta)^2 m$).

In any case, the key point of this bound is then the following: The dependence of our bound on f is only *logarithmic* in f . It is also easy to see from the bound that the sample complexity m is also logarithmic in f . As discussed in the Introduction, this means that, from an information-theoretic point of view, one may *square* the number of features (for example by adding all cross-terms between all features), and expect to need only *twice* as much training data. We believe that this, if even only approximately realizable by search algorithms, may have tremendous consequences for feature design – that modulo computational expense, overly careful human design of features would often be unnecessary, so long as additional training data can be obtained reasonably cheaply.

To close this section, we informally restate our theoretical results in terms of upper bounds on sample complexity, if the target concept is well represented by some small number r^* of features. That is, we want the number m^* of examples required so that generalization error will be close to that of the optimal hypothesis that uses r^* features. (Slightly more formally, we want, for any fixed $\epsilon > 0$, that $\epsilon(\hat{h}) < \epsilon_g(r^*) + \epsilon$ with high probability, and where dependence of m^* on ϵ will again be hidden by the $O(\cdot)$ notation.) From the earlier theorems, it is not difficult to derive the following (upper bounds on) sample complexity:

algorithm	m^*
No feature selection	$O(f_{VC})$
STANDARD-WRAP	$O(r_{VC}^* + f^\alpha), \alpha \leq 1$
ORDERED-FS	$O(r_{VC}^* + r^* \log f)$

Particularly if r_{VC} grows superlinearly in r , we easily see STANDARD-WRAP has a significantly smaller sample complexity than not performing feature selection if $r^* \ll f$. This appears to us to be rather strong theoretical justification for performing feature selection, thereby answering the question of “why feature selection” raised in the Introduction. Also, when $r^* \ll f$, ORDERED-FS, which has sample complexity logarithmic in f , is likely to learn with many fewer training examples than STANDARD-WRAP.

4 Experimental Results

Our theoretical results predicted ORDERED-FS to be much more tolerant to having a large number of irrelevant features than STANDARD-WRAP. To test this hypothesis, we ran both algorithms on a small, artificial feature selection problem.

The learning algorithm used was logistic regression [McCullagh and Nelder, 1989], used to fit a linear discriminant function, and which, while not minimizing training error, approximates that reasonably. The input space was $X = \mathcal{R}^f$, and the first target concept c we used had only one relevant feature:

$$c(x) = \begin{cases} 1 & \text{if } x_1 + 0.2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

Training examples were corrupted at a noise rate $\eta = 0.3$, and all input features were *i.i.d.* zero-mean unit variance normally distributed random variables. The search heuristic was beam search/forward search (starting out with the empty set of features, and incrementally adding features until we have the full set of features). Forward search is a popular choice that appears to usually do well [Miller, 1990], and beam search, with a beam width of 50 in our case, should be a strict improvement. (Notice also that, while ORDERED-FS was originally formulated as consisting of $f + 1$ separate searches, it is probably most naturally implemented as carrying out all the searches “together”; our beam search implementation, which starts from zero features and incrementally considers higher numbers of features, is one example of such.) Unlike many “real life” problems, all of our input features are independent, and so there were, for example, no complicated interactions between them that could complicate the search procedure. For STANDARD-WRAP, we are searching for a feature set F so that training on $S'|_F$ would give low hold-out error. For ORDERED-FS, we are searching, for each r , for a feature set F of size r so that training on $S'|_F$ gives low training error. In the rest of this section, we will not distinguish between the “idealized” versions of these two algorithms and the approximate versions of the algorithms. All experimental results reported here are averages of 200 independent trials.

For both algorithms, the hold-out fraction γ is a parameter that had to be chosen. The analysis of [Kearns, 1996] suggests that, for a wide range of hold-out testing applications, $\gamma \approx 0.3$ is a good choice (though it is unclear STANDARD-WRAP would fall into his framework). Using this as an initial choice for γ ,

we obtain Figure 1, as we vary the total number of features. We see from the graph that ORDERED-FS is performing significantly better on this domain. For reference, the performance of learning without feature selection, using all the features and not saving any data for hold-out testing, has also been plotted; for this problem, this is not really a competitive algorithm (and it is only very slightly competitive on the other target concept we test), and we omit it from the rest of our graphs.

Earlier, our bound had predicted that as f increases, the dominating factor for the error of STANDARD-WRAP comes from testing 2^f hypotheses on γm hold-out samples, thereby possibly “overfitting” the hold-out data. For STANDARD-WRAP, it is therefore natural to see if increasing the hold-out fraction γ might alleviate this effect. Doing so, we obtain Figure 2, which shows results for STANDARD-WRAP using $\gamma = 0.3, 0.5$, and 0.7 . While still inferior to ORDERED-FS, the choice of $\gamma = 0.5$ does appear to give better performance for large f , and for the remainder of our experimental results, we report results using STANDARD-WRAP with $\gamma = 0.3$ and 0.5 .

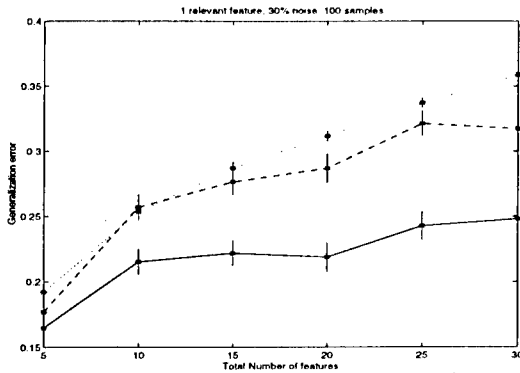


Figure 1: performance of no feature selection training on all the data (dot), of STANDARD-WRAP (dash) with $\gamma = 0.3$ and ORDERED-FS (solid) with $\gamma = 0.3$. Vertical dashes are 1se.

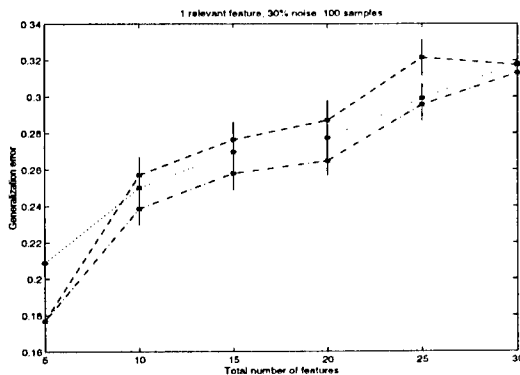


Figure 2: performance of STANDARD-WRAP using $\gamma = 0.3$ (dash), $\gamma = 0.5$ (dot-dash) and $\gamma = 0.7$ (dot). Vertical dashes are 1se.

Next, as we vary m , keeping the total number of features at 20, Figure 3 shows ORDERED-FS still consistently beating STANDARD-WRAP. Lastly, performing similar experiments with a new target function, this time with 3 relevant features

$$c(x) = \begin{cases} 1 & \text{if } x_1 + x_2 + x_3 > 0 \\ 0 & \text{otherwise} \end{cases}$$

we obtain Figures 4 and 5, which both show ORDERED-FS performing significantly better.

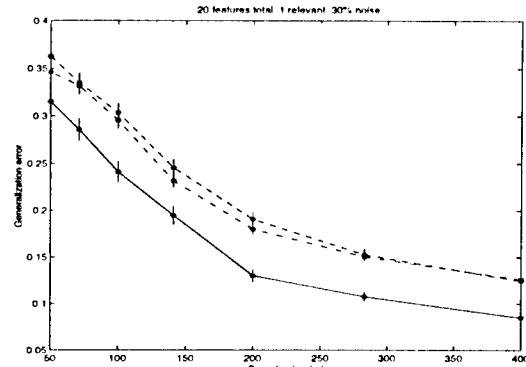


Figure 3: performance of STANDARD-WRAP with $\gamma = 0.3$ (dash) and $\gamma = 0.5$ (dot-dash), and ORDERED-FS with $\gamma = 0.3$ (solid).

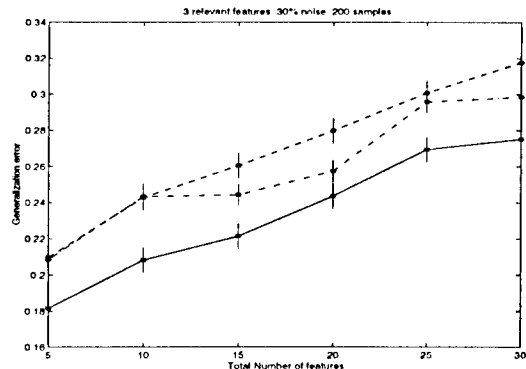


Figure 4: performance of STANDARD-WRAP and ORDERED-FS. Target has 3 relevant features. (Same legend as Figure 3.)

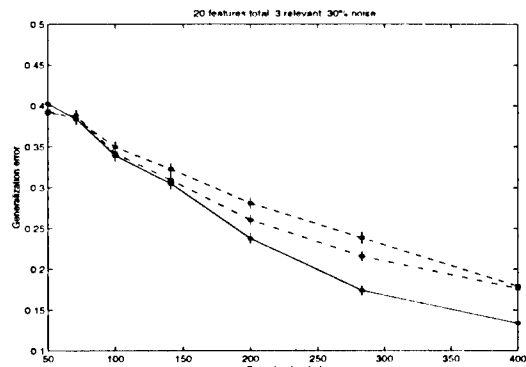


Figure 5: performance of STANDARD-WRAP and ORDERED-FS. Target has 3 relevant features.

5 Discussion and Conclusions

Our experimental results showed our heuristic-search version of ORDERED-FS generally beating that of STANDARD-WRAP. Of course, we do not claim that this will always be the case; indeed, a more detailed analysis than we had given suggests STANDARD-WRAP might do slightly better than ORDERED-FS when the number of relevant features is large, for example if $r \approx 0.5f$. (But then, this is often also the case when feature selection is less useful, compared to learning on the entire set of features.)

Throughout the paper, we skirted the issue of computational expense in (approximately) finding the best (in the training or hold-out error sense) set of features. Indeed, we believe that much work remains to be done on this field, perhaps particularly in designing algorithms for finding feature subsets that minimize training error such as ORDERED-FS requires; for example, we have very efficient algorithms for performing forward and backward search for linear regression [Miller, 1990], but few generalizations or fast approximations thereof to other algorithms. Moreover, for our bounds to predict actual performance well on real problems, we have to rely on these heuristics to perform well, though rigorous bounds for performance using search heuristics can also be given if we can bound how well the heuristic performs the required search/optimization. In particular, if heuristic approximation to STANDARD-WRAP finds only a feature subset that comes within only ε_+ of minimizing hold-out error, then a rigorous bound for its generalization error is the same as for STANDARD-WRAP with an additional ε_+ term. For ORDERED-FS, if for each value of r , we succeed in finding only a feature subset that comes within $\varepsilon_+(r)$ of minimizing training error over all feature subsets of size r , then a rigorous bound for generalization error is the same as for ORDERED-FS but with an additional $\varepsilon_+(r)$ term in the $\{\}$ curly brackets. (We defer proofs and a more detailed discussion of implications to the full paper.) Nevertheless, search heuristics are then immediately seen to be trying to drive ε_+ or $\varepsilon_+(r)$ to zero, and can therefore be argued to be trying to reach the performance suggested by our bounds. (However, one other surprising effect not modeled by our bounds and which deserves mention is that when STANDARD-WRAP is "badly" overfitting the hold-out data, then our earlier work suggests that even randomly throwing some subset of the 2^f hypotheses away may improve performance [Ng, 1997]. This suggests that in such somewhat-degenerate cases, using a weaker search heuristic may actually be helpful. In our

experiments, we did manage to find parameter ranges that seemed to exhibit this effect; but, we do not know how prevalent this effect is in practice, and would of course recommend using a good optimization criteria, like ORDERED-FS's, rather than using a less-sound criteria and then to trying to do a poor job in optimizing it.)

Finally, using techniques similar to those used in this paper, it is possible to derive other algorithms or modified versions of our algorithm that, like ORDERED-FS, have strong theoretical properties regarding tolerance to the presence of many irrelevant features, and which may have slightly different strengths and weaknesses than ORDERED-FS; and we discuss a number of them in detail in the full paper. But for now, a significant result of this work is that with appropriate feature selection, sample complexity becomes *logarithmic* in the number of irrelevant features, so that we can handle *exponentially* many irrelevant features as training examples. Of course, we still have to rely on search heuristics to help us reach these bounds, and while much empirical work remains to be done evaluating ORDERED-FS and comparing it to STANDARD-WRAP and possible interpolations between the two algorithms, we also believe that being able to give these bounds is very encouraging, because it means that if they are even only approximately realizable by search algorithms, they may have tremendous consequences for feature design – that modulo computational expense, overly careful human design of features may often be unnecessary, so long as additional training data can be obtained reasonably cheaply.

Acknowledgments

I give warm thanks to Michael Jordan and Dana Ron for interesting and helpful conversations. Also, this work would not have been possible if not for numerous greatly edifying early conversations I had with Michael Kearns about VC theory in general and related work. The author was supported by the National Science Foundation under Contract No. ASC-92-17041.

References

- [Almuallim and Dietterich, 1994] Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305.
- [Caruana and Freitag, 1994] Caruana, R. and Freitag, D. (1994). Greedy attribute selection. In *Proceed-*

- ings of the *Eleventh International Conference on Machine Learning*. Morgan Kaufmann.
- [Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman.
- [John et al., 1994] John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann.
- [Kearns, 1996] Kearns, M. J. (1996). A bound on the error of Cross Validation using the approximation and estimation rates, with consequences for the training-test split. In *Advances in Neural Information Processing Systems 8*, pages 183–189. Morgan Kaufmann.
- [Kearns et al., 1997] Kearns, M. J., Mansour, Y., Ng, A. Y., and Ron, D. (1997). An experimental and theoretical comparison of model selection methods. *Machine Learning Journal*, 27(1):7–50.
- [Kearns and Ron, 1997] Kearns, M. J. and Ron, D. (1997). Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory*. Morgan Kaufmann.
- [Kivinen and Warmuth, 1994] Kivinen, J. and Warmuth, M. K. (1994). Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, Univ. of California Santa Cruz, Computer Research Laboratory.
- [Kohavi and John, 1997] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324.
- [Kohavi and Sommerfield, 1995] Kohavi, R. and Sommerfield, D. (1995). Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*.
- [Langley, 1994] Langley, P. (1994). Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*. AAAI Press.
- [Littlestone, 1988] Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318.
- [McCullagh and Nelder, 1989] McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models (second edition)*. Chapman and Hall.
- [Miller, 1990] Miller, A. J. (1990). *Subset Selection in Regression*. Chapman and Hall.
- [Moore and Lee, 1994] Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In *Proceedings of the 11th International Conference on Machine Learning*.
- [Ng, 1997] Ng, A. Y. (1997). Preventing “overfitting” of Cross-Validation data. In *Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann.
- [Vapnik, 1982] Vapnik, V. N. (1982). *Estimation of dependencies based on empirical data*. Springer Verlag.
- [Vapnik and Chervonenkis, 1971] Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280.
- [Yang and Hoavar, 1997] Yang, J. and Hoavar, V. (1997). Feature subset selection using a genetic algorithm. In *IEEE Expert (Special Issue on Feature Transformation and Subset Selection)*. In press.

On the power of Decision Lists

Richard Nock
LIRMM
161, rue Ada
34392 Montpellier, France
nock@lirmm.fr

Pascal Jappy
LIRMM
161, rue Ada
34392 Montpellier, France
jappy@lirmm.fr

Abstract

This paper addresses the problem of using decision lists for building machine learning algorithms. In this work, we first highlight the expressive power of decision lists, which were already known to generalize decision trees. We also present ICDL, a new algorithm for learning simple Decision Lists. This problem -learning low size and high accuracy lists- is, as we prove formally, theoretically hard and calls for the use of heuristics such as CN2, BruteDL or ICDL. Our method is based on an original technique midway between learning rule based procedures and decision trees. ICDL operates in two stages : it first greedily builds a large decision list then prunes it to obtain a smaller yet accurate one, thereby avoiding the drawbacks associated with the first phase alone. Experimental results show the efficiency of our approach by comparing them to the two well-known algorithms CN2 and C4.5. ICDL's time complexity is low. It produces decision lists whose size is far smaller compared to both CN2 and C4.5, and whose accuracy also compares favourably with theirs.

1 Introduction

A Decision List (DL) is an ordered list of conjunctive rules [Riv87]. It classifies examples by assigning to each the class associated with the first rule the example triggers. Decision lists were first introduced by [Riv87], and shown to be very expressive. A motivation for the study of decision lists was their relationships with decision trees, which are widely used

as concept representations in state-of-the-art machine learning algorithms such as CART [BFOS84] or C4.5 [Qui93]. More precisely, [Riv87] proved that decision lists generalize decision trees, which proves their expressive power. In this paper we first give further insight on this property. We show that while it strictly generalizes decision trees, the decision list formalism can be used to capture the expressive power of decision committees [NG95], a class generalizing multilinear polynomials [Noc98]. Moreover, decision lists, unlike decision trees, represent classification procedures based on rules [CN89], and not on some ordering of variables. These two properties make decision lists desirable for mining sets of examples or databases, analyze their information, and improve prediction accuracy. These goals are important as many organizations tend to have massive amounts of data, which they need to understand, interpret and extrapolate [KSD96].

The work of [SE94, KSD96, Koh95, CN89, CB91] shows that any machine learning algorithm should meet four essential requirements to be of practical use:

1. Accurate classification. The induced decision list should be able to classify new examples accurately.
2. Noise handling. The algorithm should work even in domains where there might be noise.
3. Simple decision lists. For the sake of interpretability, the induced decision list should be as simple as possible. This constraint conflicts with the accuracy constraint. Generally, satisfying both implies finding a good tradeoff between simplicity and goodness-of-fit [NG95]. In practice this implies releasing the goal of finding a decision list consistent with the dataset used to build it.

4. **Efficient rule generation.** In order to handle large datasets, the algorithm must be fast. [CN89] argue that the time taken to generate one rule in the decision list should be linear in the size of the dataset used to build the decision list.

There are very few available theoretical results allowing to conclude positively to the possibility or impossibility of meeting the four quality requirements stated above. It is not even known whether finding simple and accurate decision lists is feasible within a reasonable time [dCGG94]. Yet a positive result would prove the existence of efficient algorithms for this task, and a negative result would formally prescribe the use of heuristics to meet these criteria. Our second contribution in this paper is to show that meeting the four requirements above for decision lists is hard. As has previously been done for decision trees [HR76], we prove that finding the smallest decision list consistent with a set of examples is *NP-Hard*, for various notions of sizes. Keeping in mind the four requirements mentioned above, this justifies the heuristic character of algorithms such as CN2 [CN89], BruteDL [SE94], SDL [dCGG94], or ICDL, the algorithm we propose below.

There are at least three categories of algorithms that learn decision lists, each following a different construction method. The first are greedy, iterative algorithms such as CN2 [CN89] which add rules one-at-a-time in the decision list according to a quality criterion. When a rule is built, the examples it covers are removed from the training set. The process is repeated until a stop criterion is satisfied (*e.g.* the dataset is exhausted). The second are based on a search of the rule space to find a set of good rules, before putting them into a decision list, in the same way as BruteDL [SE94]. The search is based on a branch-and-bound algorithm and proceeds by specializing iteratively a set of rules initialized to the empty set. The aim is to find all the most general homogeneous rules, that is, those rules whose accuracy does not change when specialized. The third are based on a stochastic search of the decision list space, as SDL does with simulated annealing [dCGG94].

However, practical shortcomings have been observed in all these families. As pointed out by [SE94], algorithms such as CN2 suffer from the *rule overlap* problem. When large decision lists are greedily constructed, the last rules in the decision list are built using very few examples, a situation which has two consequences: these rules are difficult to comprehend

and they may exhibit low accuracy w.r.t. new examples. The problem with algorithms such as BruteDL is that the search of the rule space that can take too long: this requires the use of thresholds to limit the search [SE94], and eventually leads to the construction of very large decision lists. Finally, algorithms such as SDL suffer the problem of stochastic search convergence, making it necessary to run the algorithm for long periods, a situation which makes them potentially time consuming.

Our third contribution in this paper is a new algorithm for the induction of short decision lists. ICDL, which stands for “Induction of CART-based Decision Lists”, is a two-stage heuristic. First, it proceeds by building rules iteratively and greedily using a procedure inspired from decision tree induction. ICDL then prunes the decision list using a CART-like criterion to obtain a shorter decision list used for testing. ICDL takes advantage of the adaptation to decision lists formalism of previous successful approaches for building decision trees.

The rest of this paper is organized as follows. First we give results completing those of [Riv87] on the expressive power of decision lists. Then we give formal proof of the hardness of building small and accurate decision lists. In the following section we present ICDL, adducing experimental results which prove the validity of our approach w.r.t. the four introductory requirements. We then compare our results with those obtained using CN2 and C4.5.

2 Expressive power of Decision Lists

Following [SE94], we let E denote the universe of examples, each of which is described using n variables. In this section, we consider for simplicity that each variable is *Boolean*. Given a set of n Boolean variables, we let $\{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$ denote the set of corresponding literals. We suppose that examples are classified according to a set of goal classes denoted G . We note a rule: $t \rightarrow g$, where $g \in G$, and t is a non-empty *monomial*, that is, a conjunction of literals. We note a decision list with k rules: $(t_1 \rightarrow g_1), (t_2 \rightarrow g_2), \dots, (t_k \rightarrow g_k), g_{k+1}$ where the class g_{k+1} is called the default class. The class associated to any example $e \in E$ is the goal class corresponding to the first monomial satisfied by the example. If none is passed, the example is assigned the default class. $\forall 0 < k < n$, we let k -DL denote the

set of decision lists whose monomials have at most k literals. Let DT stand for the class of binary decision trees, as described *e.g.* in [Riv87]. Let k - DT be the set of decision trees having depth at most k , which is the size of the longest path from the root to a leaf of the tree. In his seminal article on DL , [Riv87] proved how decision lists generalize decision trees, by studying the inclusion relationships between classes k - DL and k - DT . More formally, we have

Theorem 1 [Riv87] $\forall 0 < k < n, k\text{-}DT \subset k\text{-}DL$.

[NG95] have presented a new class of Boolean formalism allowing to precise the real power of decision lists : the *decision committees*. A decision committee contains two parts: a set of unordered couples (or rules) $\{(t_i, \vec{v}_i)\}$, where each t_i is a monomial over $\{0, 1, *\}^n$ and each \vec{v}_i is a vector in \mathbb{R}^c (in the two-classes case, it is sufficient to add a single number rather than a 2-component vector). It also contains a Default Vector \vec{D} in $[0, 1]^c$. Again, in the two-classes case, the reader shall remark that \vec{D} can be replaced by a default class $\in \{+, -\}$.

The classification of any example $e \in E$ is made by summing in a vector \vec{V}_e the vectors of each rule e satisfies. If the maximal component of \vec{V}_e is unique then its index gives the class assigned to e . Otherwise, we take the index of the maximal component of \vec{D} corresponding to the maximal components of \vec{V}_e . Let DC stand for the whole class of decision committees. Define $\forall k \leq n$, k - DC to be the subclass of DC where each element has monomials of length $\leq k$. The following theorem shows that decision committees are at least as expressive as decision lists:

Theorem 2 [NG95] $\forall 0 < k < n$ constant, $k\text{-}DL \subset k\text{-}DC$.

Although theorem 1 still holds for non-constant value of k , we have $(n-1)\text{-}DL = (n-1)\text{-}DC$ [Noc98]. This states that there exist a value k' such that $\forall k > k'$, classes k - DL and k - DC coincide, although k - DL still strictly contains k - DT . Apart from the fact that decision lists are rule-based procedures unlike decision trees, this result is another reason to advocate for use of decision lists instead of decision trees in machine learning algorithms.

3 Learning small accurate DLs

In the introduction, we have presented four requirements which should try to meet any efficient learning algorithm. Previous studies on decision trees [HR76]

and decision committees [NG95] have established that they are hard to satisfy for machine learning algorithms using these respective classes. These results justify a part of the heuristic nature of algorithms such as CART [BFOS84], C4.5 [Qui93], or IDC [NG95]. We now prove that this aim is also intractable for DL and thereby offer a positive answer to a conjecture of [dCGG94]. We define the size of a DL as the total number of literal occurrences in the decision list (if a literal appears k times, it is counted k times). This definition is very close to the one used in [Ris78] which is the smallest number of bits needed to write down a procedure, given an optimal encoding.

Theorem 3 *It is NP-Hard to find the smallest decision list consistent with a set of examples LS .*

Proof: We use a reduction from the NP-Hard "Minimum Cover" problem [GJ79]:

- **Name** : "Minimum Cover".
- **Instance** : A collection C of subsets of a finite set S . A positive integer K , $K \leq \text{Card}(C)$, where $\text{Card}(\cdot)$ denotes the cardinality.
- **Question** : Does C contain a cover of size at most K , that is, a subset $C' \subseteq C$ with $\text{Card}(C') \leq K$, such that any element of S belongs to at least one member of C' ?

The reduction is constructed as follows : from a "Minimum Cover" instance we build a learning sample LS such that if there exists a cover of size $\text{Card}(C') \leq K$ of S , then there exists a decision list with $\text{Card}(C')$ literals consistent with LS , and, reciprocally, if there exists a decision list with k literals consistent with LS , then there exists a cover of size k of S . Hence, finding the smallest decision list consistent with LS is equivalent to finding the smallest K for which there exists a solution to "Minimum Cover", and this is intractable if $P \neq NP$.

Let c_j denote the j^{th} element of C , and s_j the j^{th} element of S . We define a set of $\text{Card}(C)$ Boolean variables $\{v_1, v_2, \dots, v_{\text{Card}(C)}\}$, in one to one correspondence with the elements of C , which we use to describe the examples of LS . The corresponding set of literals is denoted $\{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_{\text{Card}(C)}, \bar{x}_{\text{Card}(C)}\}$. Our reduction uses two classes, one positive and one negative respectively denoted by "1" and "0". The sample LS contains two disjoint subsets : the set of positive examples LS^+ , and the set of negative ones LS^- . LS^+ contains $\text{Card}(S)$ examples, denoted by

$\{e_1^+, e_2^+, \dots, e_{\text{Card}(S)}^+\}$. We construct each positive example so that it encodes the membership of the corresponding element of S in the subsets of S present in C . More precisely, $\forall 1 \leq i \leq \text{Card}(S)$, $e_i^+ = \left(\bigwedge_{j:s_i \in c_j} x_j\right) \wedge \left(\bigwedge_{j:s_i \notin c_j} \bar{x}_j\right)$. LS^- contains a single negative example, defined by: $e^- = \bigwedge_{j=1}^{\text{Card}(C)} \bar{x}_j$.

- Suppose there exists a cover C' of S satisfying $\text{Card}(C') \leq K$. We create a single rule decision list: $(t \rightarrow 0), 1$. t equals $\bigwedge_{j:c_j \in C'} \bar{x}_j$. Since any element of S belongs to an element of C' , no positive example passes the monomial t . Thus all positive examples are correctly classified, as well as the negative example, which satisfies t . This decision list contains $\text{Card}(C')$ literals, and is consistent with LS .

- Suppose there exists a DL h with k literals consistent with LS . Name it $(t_1 \rightarrow g_1), (t_2 \rightarrow g_2), \dots, (t_{k'} \rightarrow g_{k'}), g_{k'+1}$, with $k' \leq k$. The three properties below hold because h is consistent.

[P1] Since there exists only one negative example, we can suppose without loss of generality that only one goal class is negative. This is either the default class if the negative example satisfies no monomial inside the decision list, or a goal class from a rule whose monomial is satisfied by the negative example. Any other goal class is positive. Let g_l denote the goal class (or the default class) which is negative, where l is an integer from the set $\{1, 2, \dots, k' + 1\}$.

[P2] Any rule preceding g_l in h contains a literal involving an equality comparison to "1". Otherwise, the negative example would satisfy the corresponding monomial, thereby being incorrectly classified.

[P3] If g_l is not the default class, monomial t_l is not empty and contains only negative literals. Otherwise, the negative example would not pass the monomial.

We now create two subsets of C , namely C'_1 and C'_2 , whose union $C' = C'_1 \cup C'_2$ is a cover of S with at most k subsets.

$$C'_1 = \begin{cases} \emptyset & \text{if } l = 1 \\ \{c_i : \exists 1 \leq j < l, x_i \in t_j\} & \text{if } l > 1 \end{cases}$$

If g_l is not the first goal class, C'_1 contains all indices of positive literals appearing in rules $t_j \rightarrow c_j$ with $j < l$.

$$C'_2 = \begin{cases} \emptyset & \text{if } l = k' + 1 \\ \{c_i : \bar{x}_i \in t_l\} & \text{if } l < k' + 1 \end{cases}$$

C'_2 contains all indices of literals variables appearing in t_l , if g_l is not the default class.

At least one of the two subsets is not empty. Otherwise, that would mean $l = 1 = k' + 1$, leading to

$k' = 0$: the decision list would consist in one default class for all negative and positive examples, which is impossible.

Because the DL is consistent, any positive example either satisfies a monomial t_m before t_l , or does not satisfy monomial t_l . In the first case, property [P2] implies that the example has some positive literal in common with t_m ; therefore, one element in C'_1 contains the element of S from which the positive example was created. In the second case, if the positive example does not satisfy monomial t_l , then by property [P3], there is in t_l at least one negative literal it does not satisfy. To this negative literal in t_l corresponds a positive one in the example, and therefore there exists an element of C'_2 containing the element of S from which the positive example was created. The union $C'_1 \cup C'_2 = C'$ contains at most k elements, and is a cover of S . This achieves the proof. \square

The limitation of the whole number of literals of a decision list is one of the finest size notion, since it comes close to the one of [Ris78]. However, the problem of constructing decision lists with limited complexity is also hard for a relaxed notion of size presented in [KLPV87] : the whole number of rules.

Theorem 4 [Noc98] *It is NP-Hard to find the smallest decision list consistent with a set of examples LS , if the notion of size is the number of rules of the decision list.*

4 ICDL

We now present our two-stage, decision list learning algorithm, ICDL. The first stage consists of the greedy construction of a large decision list, dl_{\max} , and the second one prunes dl_{\max} , to obtain dl_{end} , the decision list used for testing.

4.1 Building dl_{\max}

Table 1 presents a pseudo-code description of the algorithm used to build dl_{\max} , as well as two procedure it uses, `MakeRule()` and `BestL()`. Function `Gini()` returns the value of the Gini criterion [BFOS84] of a decision list in the following way. Let $(t_1 \rightarrow g_1), (t_2 \rightarrow g_2), \dots, (t_k \rightarrow g_k), g_{k+1}$ denote the decision list `CurrentDL`. $\forall 1 \leq i \leq k + 1, \forall 1 \leq j \leq \text{Card}(G)$, let $LS_{i,j} \subset LS$ stand for the subset of examples from class j that are classified by goal class g_i (which is the default class if $i = k + 1$) ; $\forall 1 \leq i \leq k + 1$, define

```

BuildLmax()
  DecisionList := MakeEmptyDL();
  StopDLConstruction := FALSE;
  WHILE StopDLConstruction = FALSE DO
    CurrentRule := MakeRule(DecisionList);
    IF NotEmpty(CurrentRule) THEN
      AddLast(CurrentRule, DecisionList);
    ELSE StopDLConstruction := TRUE;
  END
END

MakeRule(CurrentDL)
  newRule := MakeEmptyRule();
  StopRuleConstruction := FALSE;
  WHILE StopRuleConstruction = FALSE DO
    LTest := BestL(CurrentDL, newRule);
    IF NotEmpty(LTest) THEN
      AddLiteral(LTest, newRule);
    ELSE StopRuleConstruction := TRUE;
  END
  Return(newRule);
END

BestL(CurrentDL, Rule)
  newDL := CurrentDL;
  newRule := Rule;
  GiniOpt := Gini(CurrentDL);
  optimall := MakeEmptyLiteral();
  FOR LTest := FirstLTest to LastLTest DO
    AddLiteral(LTest, newRule);
    AddLast(newRule, newDL);
    IF Gini(newDL) < GiniOpt THEN
      optimall := LTest;
      GiniOpt := Gini(newDL);
    newDL := CurrentDL;
    newRule := Rule;
  END
  Return(optimall);
END

```

Table 1: Pseudocode for the building of dl_{max} .

```

PruneDL(DecisionList)
  DLSequence := MakeEmptySequence();
  CurrentDL := DecisionList;
  WHILE NotEmpty(CurrentDL) DO
    DLSequence := DLSequence + CurrentDL;
    CurrentLit := LitToPrune(CurrentDL);
    Prune(CurrentDL, CurrentLit);
  END
  ReturnBestDL(DLSequence);
END

```

Table 2: Pseudocode for pruning dl_{max} to obtain dl_{end} .

$LS_i = \cup_{j=1}^{Card(G)} LS_{i,j}$. $\forall 1 \leq i \leq k+1$, define

$$Gini(i) = \sum_{j \neq k}^{Card(G)} \frac{Card(LS_{i,j})}{Card(LS_i)} \times \frac{Card(LS_{i,k})}{Card(LS_i)}$$

The Gini criterion measured for CurrentDL equals

$$Gini(CurrentDL) = \sum_{i=1}^{k+1} \left(\frac{Card(LS_i)}{Card(LS)} \times Gini(i) \right)$$

BuildLmax() adds rules at the end of a decision list initialized to the empty decision list, using the procedure AddLast(). Each rule is constructed using MakeRule(). This construction consists in building a rule by adding literals one-at-a-time using the procedure BestL(). The best literal returned by BestL() is the one, if it exists, that satisfies the two following conditions:

- it diminishes the most the Gini criterion of the whole decision list, for any addition of literal in the current rule constructed, and
- it diminishes the value of the Gini criterion compared to the value of the decision list before addition of the literal.

BuildLmax() stops when any one-literal rule added at the end of the decision list fails to lower the value of Gini criterion.

4.2 Pruning of dl_{max} to obtain dl_{end}

Table 2 presents a pseudo-code description of the algorithm used to prune dl_{max} , to obtain dl_{end} . In our experiments, the examples set used to prune is different from the one used to construct dl_{max} . At each step, the literal to be pruned is returned by the

procedure `LitToPrune()`, and is the literal l , among all those of `CurrentDL`, which minimizes the function `PruneValue(CurrentDL, l)`:

$$\text{PruneValue}(\text{CurrentDL}, l) = \frac{\text{Delta}(\text{CurrentDL}, l)}{N(\text{CurrentDL}, l) \times (\text{NDT}(\text{CurrentDL}, l) - 1)}$$

`Delta(CurrentDL, l)` returns the number of examples well classified by `CurrentDL`, and that are no longer well classified when we remove literal l . `N(CurrentDL, l)` returns the number of examples that do not pass any literal preceeding l in the decision list. `NDT(CurrentDL, l)` returns the product of sizes of all rules following the one in which l is present. This criterion is analogous to the pruning criterion used in CART [BFOS84], measured over a decision tree built from the decision list such that each node in the decision tree is one literal corresponding to an literal in the decision list. We refer the reader to [BFOS84] for additional details on this criterion. $d1_{\text{end}}$, is the DL which, among all DLs pruned from $d1_{\text{max}}$, has the highest accuracy over the examples used to prune $d1_{\text{max}}$.

5 Experimental results

ICDL was run on several benchmark problems. Its results are compared to those of CN2 [CN89] and C4.5 [Qui93]. Table 3 presents the datasets which were used for comparisons (c is a shorthand for `Card(G)`). Dataset references are [CB91] for V0, PC, GL, HH, HC, EC, HP, [TBB⁺91] for M1, M2, M3 and [BFOS84] for W0. All datasets, except W0 (artificial problem generated following [BFOS84]), can be found in the *UCI repository of Machine Learning database*.

A learning sample LS is split in two subsets, the

Table 3: Characteristics of Data Sets.

	<code>Card(LS)</code>	c	Test set	Comments
V0	435	2	0	Congress votes
W0	10×300	3	5000	Waveform recognition
M1	124	2	432	MONKS #1
M2	169	2	432	MONKS #2
M3	122	2	432	MONKS #3
PC	1044	2	0	Pole and Cart
GL	214	6	0	Glass recognition
HH	294	2	0	Heart Hungary
HC	303	2	0	Heart Cleveland
EC	131	2	0	Echocardio
HP	157	2	0	Hepatitis

first used for building $d1_{\text{max}}$ (2/3 of the examples), and the second one for pruning to obtain $d1_{\text{end}}$ (1/3 of the

Table 4: ICDL vs CN2.

	ICDL acc	CN2 acc	ICDL size	CN2 size
V0	95.9±1.0	94.8±1.7	3.1 ± 1.6	41.6±8.2
W0	66.5±4.6	65.6±4.3	21.8±6.8	28.6±5.1
M1	83.3	100	6	13
M2	65.1	69	14	145
M3	100	89.1	4	38
PC	70.7±1.5	70.6±3.1	14.8±9.8	133.6±6.3
GL	58.9±6.4	58.5±5.0	12.6±3.9	32.8±3.0
HH	79.4±3.4	75.4±3.6	13.4±6.9	35.1±2.5
HC	79.0±4.2	75.0±3.8	15.5±7.5	40.9±4.0
EC	70.7±4.7	62.3±5.1	4.7±2.4	26.4±4.0
HP	79.6±3.5	77.6±5.9	4.2±2.8	24.0±5.5
Avg.	77.19	76.17	10.37	50.81

Results for CN2 are those of [CB91] (V0, PC, GL, HH, HC, EC, HP), [TBB⁺91] (M1, M2, M3) and [dCGG94] (W0).

examples). When only one dataset exists for learning and testing (which is the case for all datasets except W0, M1, M2, M3), we proceed by averaging over 10 iterations the result of the following split-and-build experiment: randomly split the whole sample into a learning sample (2/3 of the examples) and a test sample (1/3 of the examples); use the learning sample to construct a decision list with ICDL, and test it on the test set (ratios follows [CB91]). Therefore, in that case, 4/9 of the examples are used for building $d1_{\text{max}}$, 2/9 are used for building $d1_{\text{end}}$, and 1/3 are used for evaluating the accuracy of $d1_{\text{end}}$. Table 4 presents ICDL's results compared to CN2's ("acc" means accuracy ; "size" stands for the whole number of literals in the formula ; "Avg." gives average results) ; when there are more than one learning sample (W0), or when split-and-build is used, results are given in the form "Mean±Standard deviation".

On all but two problems, ICDL achieves better accuracies than CN2. Results are even more favourable if we take into account the sizes obtained. In all datasets, the DLs found by ICDL are much smaller than those of CN2. If we exclude W0, sizes obtained for ICDL are up to fourteen times smaller than CN2's.

Comparisons with the state-of-the-art decision tree learning algorithm C4.5 are presented in table 5. Decision tree size is the number of nodes *including leaves* [CB91]. To make correct comparisons, a DL size given is now total number of literals *plus* total number of classes in the DL. With the exception of PC and GL, ICDL outperforms C4.5 on all datasets. Again, the size comparison points out important differences, that are on average in favor of ICDL. However, the gap is less important than for CN2 and on three problems,

Table 5: ICDL vs C4.5.

	ICDL acc	C4.5 acc	ICDL size	C4.5 size
V0	95.9±1.0	95.6±1.1	6.9 ± 2.7	7.7±3.4
M1	83.3	80.6	10	
M2	65.1	64.8	23	
M3	100	97.2	8	
PC	70.7±1.5	74.3±3.1	22.6±13.2	90.2±10.2
GL	58.9±6.4	64.2±5.1	18.9±5.2	30.9±5.8
HH	79.4±3.4	78.0±5.5	21.7±10.0	7.2±3.7
HC	79.0±4.2	76.4±4.5	24.0±9.9	22.7±4.6
EC	70.7±4.7	63.6±5.3	9.1±3.7	9.2±4.7
HP	79.6±3.5	79.3±5.8	7.5±3.7	6.4±2.6
Avg.	78.26	77.4	15.81*	24.9*

Results for C4.5 are those of [CB91] (V0, PC, GL, HH, HC, EC, HP) and [Koh95] (M1, M2, M3).

(*) Average sizes do not take into account M1, M2, M3 (we did not have C4.5's).

the formulae sizes found by C4.5 are actually smaller than for ICDL.

6 Discussion

While the results above illustrate ICDL's good performances, we now discuss how close this algorithm comes to the four widely accepted requirements presented in the introduction : high accuracy, noise tolerance, small sizes, and low time complexity. As pointed out by [CN89], time complexity needs only to be evaluated on the crucial steps of the algorithm. During the construction of dl_{max} , ICDL's crucial step is the same as CN2: namely the addition of a single literal to the current rule. So this complexity is that of the `BestLiteral()` subroutine. It represents $\mathcal{O}(\text{Card}(LS) \times \text{Card}(\text{Atests}))$ in ICDL, where "Atests" denotes the set of possible literals. This complexity is smaller than that of CN2 [CN89]. The time complexity of the crucial step of the pruning phase corresponds to the complexity of the `LiteralToPrune()` subroutine, which determines which literal is to be removed from the current formula. Its time complexity is $\mathcal{O}(\text{Card}(LS) \times \text{Card}(\text{AtestsFormula}))$, where "AtestsFormula" denotes the multiset of literals of the current formula. In fact, ICDL's whole complexity is not high w.r.t. classical induction algorithms.

M2 and V0 are relevant to the discussion of sizes. In V0, empirical studies (see the ML repository) show that there exists a single literal DL that performs around 95 % accuracy. We noticed that the DL found

by ICDL always included this formula, which leads to excellent tradeoffs between size and accuracy. On the artificial domain M2, the function to learn is an XOR function [TBB⁺91], which is very difficult to encode with small DL (compare with the result obtained by CN2). Again, in this case, ICDL found a tiny formula which is highly accurate considering its size and the difficulty of the domain.

ICDL obtained very good results w.r.t. the complexity vs accuracy tradeoff. As a means of evaluating this for each possible dataset, we have calculated the accuracy/size ratio, which constitutes a rough "informative measure" of each literal with respect to the overall accuracy. Provided accuracies are sufficiently high (which was the case for CN2, C4.5 and ICDL), the higher this ratio, the better and the more interesting the accuracy/size compromise the algorithm obtains. The calculation shows that, for each dataset, this ratio is higher for ICDL than for CN2. Furthermore, with the exception of M1 and W0, ICDL's lowest ratio is higher than CN2's highest. Finally, if we exclude the HH and HC problems, ICDL also outperforms C4.5 on all domains for which we possess accuracy and size measures for C4.5. Tables 4, 5 and [CB91] show that for datasets V0, PC, GL, EC, HP, ICDL outperforms C4.5, which in turn outperforms CN2. ICDL's accuracy is on average slightly better than C4.5's, yet its average output size is smaller. Thus, ICDL appears to be able to compact the knowledge of the decision trees in small DLs. This result concords with [Riv87], section 3.2, who shows that decision lists generalize decision trees.

Finally, ICDL's handling of noise can be experimentally evaluated using problems W0 and M3, which are artificial noisy problems. On both problems, ICDL's results are good. In M3, there is a little noise, and very few learning algorithms in [TBB⁺91] achieve 100% accuracy. ICDL achieves the perfect classification, and surpasses all inductive learning algorithms tested in [TBB⁺91] : ID3, ID5R, AQR, CN2 and CLASSWEB. Again, the size of the formula found by ICDL is smaller than that of these algorithms.

[SE94] point out a limiting aspect of decision list construction using greedy algorithms such as CN2 : rules cannot be considered in isolation, and, after each rule building stage, fewer examples are available to the learning algorithm. ICDL reduces the adverse effects of both problems by building small decision lists using efficient pruning. Indeed, the pruning step uses new examples, which are not used for building dl_{max} . Furthermore, it uses a criterion reducing the effect of the

limited number of examples available to the rules at the end of the decision list.

7 Conclusion

In this paper, we introduce ICDL, a new algorithm for learning simple decision lists. Its originality stems from the adaptation to decision lists of the combination of two techniques which have proved very efficient in CART and C4.5. It combines the building of a decision list in a greedy way using a Gini criterion calculated on the whole decision list. It then prunes the decision list using a CART-like criterion. However, its result formulae are rule-based procedures which provide an alternative to decision trees for building knowledge-based systems.

We prove formally that inducing short and accurate DLs is intractable, which prescribes the use of heuristics such as CN2, ICDL, or BruteDL [SE94]. We then give experimental results which compare very favourably to those of CN2 and C4.5, and which show experimentally that ICDL meets the accepted criteria of low-time complexity, noise handling, small output size and high accuracy. We believe this efficiency is due to ICDL's successful application of a combination of decision tree learning techniques to the more expressive DL representation.

Acknowledgements

Many thanks to Ronald Rivest for having encouraged us.

References

- [BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
- [CB91] P. Clark and R. Boswell. Rule induction with CN2: some recent improvements. In *ECML'91*, pages 151–161, 1991.
- [CN89] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
- [dCGG94] F. A. de Carvalho Gomes and O. Gascuel. SDL, a stochastic algorithm for learning decision lists with limited complexity. *A. of Math. & AI*, pages 281–302, 1994.
- [GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability, a guide to the theory of NP-Completeness*. Bell Telephone Laboratories, 1979.
- [HR76] L. Hyafil and R. Rivest. Constructing optimal decision trees is NP-complete. *Inform. Process. Letters*, pages 15–17, 1976.
- [KLPV87] M.J. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. *STOC'87*, pages 285–295, 1987.
- [Koh95] R. Kohavi. The power of Decision Tables. In *ECML'95*, pages 174–189, 1995.
- [KSD96] R. Kohavi, D. Sommerfield, and J. Dougherty. Data Mining using MLC++. In *Int. Conf. on Tools with AI*, pages 234–245, 1996.
- [NG95] R. Nock and O. Gascuel. On learning decision committees. In *Proc. of the 12th International Conference on Machine Learning*, pages 413–420, 1995.
- [Noc98] R. Nock. *Learning logical formulae having limited size : theoretical aspects, methods and results*. PhD thesis, Université Montpellier II, 1998.
- [Qui93] J. R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465, 1978.
- [Riv87] R.L. Rivest. Learning decision lists. *Machine Learning*, pages 229–246, 1987.
- [SE94] R. Segal and O. Etzioni. Learning Decision Lists using homogeneous rules. In *Proc. of AAAI-94*, pages 619–625, 1994.
- [TBB⁺91] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

An Analysis of Direct Reinforcement Learning in non-Markovian Domains

Mark D. Pendrith*

Department of Artificial Intelligence
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 Australia
pendrith@cse.unsw.edu.au

Michael J. McGarity

Department of Systems and Control
School of Electrical Engineering
The University of New South Wales
Sydney 2052 Australia
mikem@cse.unsw.edu.au

Abstract

It is well known that for Markov decision processes, the policies stable under policy iteration and the standard reinforcement learning methods are exactly the optimal policies. In this paper, we investigate the conditions for policy stability in the more general situation when the Markov property cannot be assumed. We show that for a general class of non-Markov decision processes, if actual return (Monte Carlo) credit assignment is used with undiscounted returns, we are still guaranteed the optimal observation-based policies will be equilibrium points in the policy space when using the standard "direct" reinforcement learning approaches. However, if either discounted rewards, or a temporal differences style of credit assignment method is used, this is not the case.

1 Introduction

The techniques of reinforcement learning (RL) have been developed to effect autonomous learning in agents interacting with an initially unknown and possibly changing environment. In its simplest formulation, the problem of RL is cast into a table lookup representation, where the agent can be in one of a finite number of states at any time, and has the choice of finite number of actions to take from within each state. For this representation, powerful convergence and optimality results have been proven for a number of algorithms designed with the simplifying assumption that the environment is Markov, e.g. 1-step Q-learning (Watkins, 1989). With this assumption, the problem of learning can

be cast into the form of finding an optimal policy for a Markov decision process (MDP), and methods like 1-step Q-learning can be shown to be a form of incremental, asynchronous dynamic programming (Watkins, 1989; Barto, Bradtke, & Singh, 1995).

In practice, however, RL techniques are routinely applied to many problem domains for which the Markov property does not hold. This might be because the environment is non-stationary, or is only partially observable; often the side-effects of state-space representation can lead to the domain appearing as non-Markov to a reinforcement learning agent.

In this paper, we examine various issues arising from applying standard RL algorithms to non-Markov decision processes (NMDPs). In particular, we are interested in the implications of using a "direct" or *observation-based* method of RL for a non-Markov problem, i.e. where the problem is known to be non-Markov but partial or noisy state observations are presented directly to the RL algorithm without any attempt to identify a "true" Markov state (Singh, Jaakkola, & Jordan, 1994).

2 Policy stability in Dynamic Programming

In this section, we review the important idea of a stable policy in terms of classical dynamic programming (DP) methods.

It is well known (e.g. Puterman (1994)) that for any MDP, all suboptimal policies are unstable under policy iteration i.e. one step of the policy iteration process will result in a different policy. Moreover, the new policy will be a better policy; and so the process of policy iteration can be viewed as a hill-climbing process through the policy space of stationary policies, i.e. the result of each step in policy iteration results in a monotonic improvement in policy until an optimal policy is reached.

* Current address: Daimler-Benz Research and Technology Center, 1510 Page Mill Rd, Palo Alto, CA 94304, USA. e-mail: pendrith@rtna.daimlerbenz.com

Any optimal policy will have the property of being stable under a single step of policy iteration. The special properties of a Markov domain ensure the policy space to be well-suited to a hill-climbing strategy; there are no “local maxima” or suboptimal equilibrium points to contend with, and all the global maxima form a single connected “maxima plateau” that can be reached by starting a hill-climbing process from any point in the space.

It is also the case that a “partial” policy iteration, where only a subset of the states that would have policy changes under a full policy iteration step have their policy actions changed, will also monotonically improve the policy, and therefore result in effective hill-climbing. This is the key property that makes MDPs susceptible to RL techniques; it has become the convention to characterise RL in Markov domains as an incremental, asynchronous form of dynamic programming (Watkins, 1989; Barto et al., 1995). If the RL method is a 1-step temporal differences (TD) method, like Watkins’ 1-step Q-learning, the method resembles an incremental, asynchronous form of value-iteration. If the RL method is an actual return or Monte Carlo based method, like C-Trace (Pendrieth & Ryan, 1996) the method more closely resembles an incremental, asynchronous form of policy iteration.

So, for an MDP, the optimal policies correspond to the policy iteration equilibrium points in the policy space. By way of contrast, for NMDPs it is straightforward to demonstrate that suboptimal policy iteration equilibria are possible, and subsequently that policy iteration methods can fail by getting “stuck” in local maxima. Consider the NMDP in Figure 1.

Figure 1 shows an NMDP with two actions available from starting observation A, and two actions available from the successor observation B.¹ Both action 0 and action 1 from observation A immediately lead to observation B with no immediate reward. Action 0 and action 1 from observation B both immediately lead to termination and a reward; the decision process is non-Markovian because the reward depends on what action was previously selected from observation A, according to the schedule in Table 1.

In the policy space for this NMDP, the deterministic policy π_3 is clearly optimal, with a total reward of 2. Further, it represents an equilibrium under policy iteration: if states A or B independently change policy, the total reward becomes -2 and 0 respectively. Notice that policy π_0 , although clearly suboptimal with a total reward of 1 , is also

¹In general, we will be referring to the “observations” rather than “states” of an NMDP, as we will be moving on later to discuss a specific class of NMDPs that are defined in a POMDP framework, where this terminological distinction becomes important.

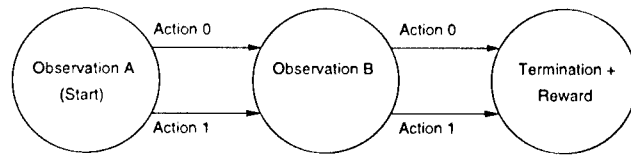


Figure 1: An NMDP demonstrating an suboptimal equilibrium.

	A action	B action	reward
π_0	0	0	1
π_1	0	1	-2
π_2	1	0	0
π_3	1	1	2

Table 1: Reward schedule for NMDP in Figure 1.

an equilibrium: if states A or B independently change policy, the total reward becomes 0 and -2 respectively.

Although we have only explicitly considered deterministic policies in the above discussion, we note that the result generalises straightforwardly to stochastic policies.

In the case of the example above the optimal policy was also a deterministic policy. However, it is known that in general for NMDPs there may be no deterministic policy among the optimal policies, as will always be the case for MDPs (Singh et al., 1994).

Further, we will show in this paper that if a TD method of credit assignment is used, or the rewards are discounted, the optimal policies may not represent equilibrium points in the policy space, even if an optimal deterministic policy exists. This means that even if the problems of local maxima are overcome, the optimal policies may not be attractive under some standard RL techniques.

It turns out the key property of optimal policies being stable under RL is only preserved if the additional restrictions of using undiscounted rewards and using actual return credit assignment methods are imposed.

3 Learning Equilibria

For the analysis of standard RL algorithms for NMDPs, it is useful for us to introduce the notion of a *learning equilibrium*, an equilibrium in policy resulting from a particular learning method. So just as we can talk about a policy that is stable under policy iteration, we might talk about a policy that is stable under 1-step Q-learning, for example.

Definition 1 A *learning equilibrium* has the property that if you replace the current state (or (state, action)) value estimates with the expected value of the those estimates given

the current policy and the learning method being used, then the policy remains unchanged.

A learning equilibrium can be considered to be a *stochastic fixed point* in the policy space with respect to a given learning method.

We consider that, in general, an RL system will in the course of learning perform a series of updates to a set of real-valued utility estimators. These estimators will typically estimate state value or $\langle \text{state}, \text{action} \rangle$ value, or sometimes both. Further, we are assuming the current policy of an RL system will be a function of these estimator values, which we might write as $f : E \rightarrow \Pi$, where E represents the space of possible estimator values, and Π is the policy space.

We can also consider a mapping in the reverse direction $g : \Pi \rightarrow E$, where the point $g(\pi)$ in the estimator space corresponds to the expected values of the estimators with respect to the learning rules and a policy π . For example, if the system is a Q-learning system, $g : \Pi \rightarrow E$ would be defined by the Q function, where $Q^\pi(s, a)$ is the expected value of the $\langle s, a \rangle$ estimator under policy π .

If we consider $h : \Pi \rightarrow \Pi$ to be the composition of functions f and g such that $h(\pi) = f(g(\pi))$, then if a policy π' meets the fixed point condition $\pi' = h(\pi')$, then π' is a learning equilibrium. In this way a learning equilibrium can be considered a generalisation of the notion of a policy that is stable under policy iteration. However, given the stochastic nature of the g mapping, such a fixed point represents stability in terms of expectation only.

For any MDP with a total discounted reward optimality criterion, the only equilibrium policies for any of the RL or DP methods discussed so far will be optimal policies. A policy that is stable under policy iteration is also stable under value iteration, or under 1-step Q-learning.

On the other hand, in a non-Markov setting there may be suboptimal equilibria for RL systems. The example in Figure 1 provides an example of this possibility.

Clearly, having a global maximum in policy space which is also a learning equilibrium is a necessary condition for convergence to an optimal policy under a given learning method. This basic idea provides the motivation for the form of analysis that follows.

4 hPOMDPs

The essence of an NMDP is that the history of states and actions leading to the present state may in some way influence the expected outcome of taking an action within

that state. When applying a standard RL method like 1-step Q-learning to an NMDP, the history is not used even if available — this is what Singh et al. (1994) call *direct* RL for NMDPs. Therefore, one potentially useful approach to modelling a general class of NMDPs is by considering a process that becomes Markov when the full history of states and actions leading to the present state is known, but may be only *partially observable* if this history is not available or only partially available, i.e. the full history is guaranteed to provide any missing state information.

Another way of expressing this is to say that nothing apart from the currently observed state information along with the history is required to provide a sufficient statistic. This property defines a class of partially observable Markov decision process (POMDP) we will call hPOMDPs (with h for history).

We should emphasise that in the hPOMDP model the full history is always sufficient, but not always necessary, to disambiguate the true state. hPOMDPs include processes where only some or even none of the history is required. For example, a fully Markov process, which requires no history at all to disambiguate the state, is included in the hPOMDP class. Another example would be a process that only requires the current observation plus the starting observation for full state disambiguation. Using a POMDP formulation, we can formalise the properties of an hPOMDP stated above by requiring the existence of a function $\phi(s, h)$ that maps the current observation s and history h into a unique state in the underlying MDP.

The original motivation behind the formalisation of hPOMDPs was to provide a model for the sort of non-Markovianity that is encountered when state aggregation due to state-space representation or other forms of state-aliasing occur; usually, in cases like these, history can make the observation less ambiguous to some extent, and the more history you have the more precisely you can determine the true state.² However, hPOMDPs may also be used to model the more discrete kinds of perceptual-aliasing more frequently encountered in the RL literature, a proto-

²We note that for some control processes, even the entire history is not able to completely disambiguate the state. For example, the original noise- and disturbance-free formulation of the pole-and-cart problem using a "boxed" state-space representation (Michie & Chambers, 1968; Barto, Sutton, & Anderson, 1983; Pendrith & Ryan, 1996) is well-modelled using hPOMDPs when the initial state of the system is known (e.g. zero for all state variables), but if the initial state variables are randomised or otherwise uncertain, then access even to the full history may not make the true current state unambiguous. However, even in this situation, we note the history will make the true state less ambiguous; and so the hPOMDP model might be considered to be a useful "limiting case" approximation for domains like these.

typical example being Kaelbling et al.'s "robot in the corridors" scenario (Kaelbling, Littman, & Cassandra, 1995).³

5 A Discounted Reward Framework for NMDPs

Because we are interested in what happens when applying standard discounted reward RL methods like Q-learning to NMDPs, we restrict our attention to the class of *finite* hPOMDPs (i.e., a hPOMDP such that the observation/action space $S \times A$ is finite).⁴ This effectively models the RL table-lookup representation for which all the strong convergence results have been proven in the context of MDPs.

5.1 Summing Over Histories

We consider a *total path* or *trace* through a finite hPOMDP which can be written as a sequence of observation/action pairs

$$(\langle s_0, a_0 \rangle, \langle s_1, a_1 \rangle, \dots, \langle s_i, a_i \rangle, \dots)$$

where $\langle s_i, a_i \rangle$ is the pair associated with the i^{th} time-step of this path through the system. For any finite or infinite horizon total path ω there is an associated total discounted reward

$$R(\omega) = \sum_{t=0}^n \gamma^t r_t \quad (1)$$

where $\gamma \in [0, 1]$ is the discount factor, r_t is the immediate reward associated with taking action a_t from observation s_t , and n is the horizon.

In measure theoretic terms, we can express the probability P_s^π of a particular observation s ever being visited under policy π as

$$P_s^\pi = P^\pi(T_s) \quad (2)$$

where the set T_s is the set of possible traces that includes s , and P^π is a suitably defined probability measure over the space of all possible traces T with respect to policy π . We can also write

$$P_{\bar{s}}^\pi = (1 - P_s^\pi) = P^\pi(T_{\bar{s}})$$

where $P_{\bar{s}}^\pi$ is the complementary probability of observation s not being visited, $T_{\bar{s}}$ being the set of traces that do not include s . These "visit probabilities" assume there is a distribution of starting observations ψ associated with an

³We note however that for such systems to be accurately modelled by hPOMDPs some additional restrictions on the problem may need to be applied, e.g. the initial state must be known to the RL agent.

⁴Note that this does *not* imply there are only a finite number of states in the underlying MDP. (cf. Singh et al., 1994).

hPOMDP, where ψ_s is the *a priori* probability of observation s being the initial observation of the process.

We note that in general, e.g. if the process is non-absorbing, a trace may be of infinite length, and therefore the associated probability of it occurring may be infinitesimal, and the set T_s uncountable; these considerations motivate introducing the techniques of measure theory.⁵

We also note that executing a trace that involves one or more visits to s is logically equivalent to executing a trace that involves a first visit to s , and therefore

$$P_s^\pi = \sum_{h \in H_s} p(h, \pi) \quad (3)$$

where H_s is the set of finite length *first-visit histories*, which are the possible chains of observation/action pairs leading to a first visit to observation s , and $p(h, \pi)$ is the associated probability of a first visit occurring by that history under policy π . Because $h \in H_s$ are of finite length, $p(h, \pi)$ is finite and H_s is countable, and therefore we can express the value as a sum rather than an integral.

The technical issue of defining an appropriate probability measure P^π consistent with the value of this sum to enable working with Lebesgue integrals is dealt with in detail in (Pendrith & McGarity, 1997), where the equivalence of (2) and (3) is used as a starting point. However, it is not necessary to immediately consider these details to follow the development of this paper.

5.2 Defining Analogs of Q-value and State Value for hPOMDPs

A stochastic policy takes the form of a set of action selection distributions, with one distribution for each observation. Thus a deterministic policy can be considered to be a special case of a stochastic policy. So for generality, we define the following hPOMDP values with respect to stochastic policies.

We consider the expected future discounted reward (i.e. utility) of taking an action randomly selected with respect to a distribution d from an observation s , with first-visit history h and following policy π thereafter. We denote this as $U^\pi(s, d, h)$. For notational convenience, we will also write $U^\pi(s, a, h)$ to represent the utility of taking a particular action a from observation s with history h and following policy π thereafter. (This can be considered shorthand where a stands in for the distribution that would deterministically select a .)

⁵For a review of the essential measure theory concepts used in this paper see e.g. (Billingsley, 1986).

We note that the values $U^\pi(s, d, h)$ and $U^\pi(s, a, h)$ are both well-defined by the definition of an hPOMDP. $U^\pi(s, a, h)$ can be considered the “Q-value” of the underlying (possibly infinite state) MDP where the action a is taken from “true” state $\phi(s, h)$. $U^\pi(s, d, h)$ would therefore be a weighted average of these Q-values for that state.

A value that is of interest if we are considering what can be learned by applying standard RL methods directly to hPOMDPs is the following weighted average of the above defined utilities

$$Q^\pi(s, d) = \begin{cases} \sum_{h \in H_s} \frac{p(h, \pi)}{P_s^\pi} U^\pi(s, d, h) & \text{if } P_s^\pi > 0 \\ \text{undefined} & \text{if } P_s^\pi = 0 \end{cases}$$

Extending our shorthand notation introduced above, we will also write

$$Q^\pi(s, a) = \begin{cases} \sum_{h \in H_s} \frac{p(h, \pi)}{P_s^\pi} U^\pi(s, a, h) & \text{if } P_s^\pi > 0 \\ \text{undefined} & \text{if } P_s^\pi = 0 \end{cases}$$

$Q^\pi(s, a)$ is what might be called the “observation first-visit Q-value”; we observe it is the value a first-visit Monte Carlo method will associate with taking action a from observation s in the hPOMDP.⁶ Similarly, $Q^\pi(s, d)$ is the expected value a first-visit Monte Carlo method will come to associate with selecting an action using distribution d from observation s .

Using the definitions above, we define the value of an observation for a policy to be $V^\pi(s) = Q^\pi(s, \pi_s)$ where π_s is the action selection distribution associated with observation s under stochastic policy π ; if π represents a deterministic policy, then π_s denotes the policy action for observation s .

We note that the values of $Q^\pi(s, d)$, $Q^\pi(s, a)$ and hence $V^\pi(s)$ are undefined for s if $P_s^\pi = 0$ (i.e., s is unreachable under π). This is because, unlike the case for MDPs, it is difficult to assign a sensible meaning to the notion of the value of taking an action from an unreachable observation. In short, the notion of an “observation first-visit Q-value” is fairly empty if a first visit simply isn’t possible.

⁶Recall that H_s is a set of first-visit histories. We consider first-visit rather than multiple-visit Monte Carlo methods because there are some basic conceptual problems with using the latter in a non-Markovian context (in the general case, it doesn’t make sense to apply multiple-visit Monte Carlo when histories may matter, i.e. the Markov assumption doesn’t hold.) For an introduction to concepts of first-visit versus multiple-visit Monte Carlo methods as applied to RL, see (Singh & Sutton, 1996).

5.3 Policy Values for hPOMDPs

We can write the policy value, or total expectation, of an hPOMDP in terms of a Lebesgue integral

$$J(\pi) = \int_{\omega \in T} R(\omega) dP^\pi(\omega) \quad (4)$$

integrating over total paths.

We can further decompose the total expectation into a conditional expectation component that involves observation s and another that is independent of change to the policy for observation s in the following expression:

$$J(\pi) = \int_{\omega \in T_s} R(\omega) dP^\pi(\omega) + \int_{\omega \in T_{\bar{s}}} R(\omega) dP^\pi(\omega) \quad (5)$$

Note that for a general discounted reward structure we can write

$$\int_{\omega \in T_s} R(\omega) dP^\pi(\omega) = \sum_{h \in H_s} p(h, \pi) [R(h) + \gamma^{l_h} U^\pi(s, \pi_s, h)] \quad (6)$$

where $0 \leq \gamma \leq 1$ is the discount factor, l_h is the length of history h , and $R(h)$ is the value of the truncated discounted return associated with history h (cf. Equation (1)). Thus, the LHS and RHS of this identity are different expressions for the conditional expectation assuming a visit to observation s .

Finally, we define an optimal observation-based policy π^* simply by

$$\pi^* \in \arg \max_{\pi} J(\pi) \quad (7)$$

These definitions provide a framework for analysing hPOMDPs using a total future discounted reward criterion which applies equally well to both ergodic and non-ergodic systems.

6 Analysis of Observation-Based Policy Learning Methods for hPOMDPs

The first result we present is a lemma useful in the proof of Theorem 1. The proof of the lemma is omitted for space reasons; for the proof see (Pendrith & McGarity, 1997).

Lemma 1 *If two observation-based policies π and $\tilde{\pi}$ for an undiscounted hPOMDP differ only in one observation s , then the difference in values between the policies π and $\tilde{\pi}$ can be expressed as*

$$J(\tilde{\pi}) - J(\pi) = P_s^\pi [V^{\tilde{\pi}}(s) - V^\pi(s)] \quad (8)$$

We note Lemma 1 has a strong intuitive basis, suggesting its applicability to a very general class of decision processes including but not limited to hPOMDPs. Equation (8) corresponds to the straightforward observation that for an undiscounted reward process, by changing policy in exactly one reachable state under policy π , the change in value of the expected total reward for the new policy is equal to the change in first-visit expected value for the changed state multiplied by the *a priori* probability that state will have a first-visit under policy π .

Theorem 1 *If a first-visit Monte Carlo method of credit assignment is used for an hPOMDP where $\gamma = 1$, then the optimal observation-based policies will be learning equilibria.*

Proof Suppose an optimal observation-based policy π is not a learning equilibrium under a first-visit Monte Carlo credit assignment method; then there must exist an observation s such that $V^{\hat{\pi}}(s) > V^{\pi}(s)$ for some policy $\hat{\pi}$ that differs from π only in observation s . By Lemma 1, the difference in policy values is

$$J(\hat{\pi}) - J(\pi) = P_s^{\pi}[V^{\hat{\pi}}(s) - V^{\pi}(s)]$$

Since $V^{\hat{\pi}}(s) > V^{\pi}(s)$ and $P_s^{\pi} > 0$ (i.e. observation s is reachable under π),⁷ then $J(\hat{\pi}) > J(\pi)$. But this is not possible since π is an optimal policy; hence an optimal policy is a learning equilibrium. \square

Theorem 1 is a positive result: it shows that, at least under certain restricted conditions, an optimal observation-based policy is also guaranteed to represent a policy equilibrium for a direct RL style learner.

The next question is whether we can generalise the result. Does the result hold for general γ ? Does the result hold for TD returns instead of Monte Carlo style “roll-outs”?

The next result addresses the issue of using discounted returns for general γ :

Theorem 2 *Theorem 1 does not generalise to $\gamma \in [0, 1)$.*

Proof We prove this by providing a counter-example. We consider the hPOMDP in Figure 2.

Figure 2 shows an hPOMDP with one action available from the two equiprobable starting observations A and B; one

⁷Note that observation s must be reachable under both π and $\hat{\pi}$ otherwise both $V^{\pi}(s)$ and $V^{\hat{\pi}}(s)$ would be undefined, which is incompatible with the hypothesis $V^{\hat{\pi}}(s) > V^{\pi}(s)$.

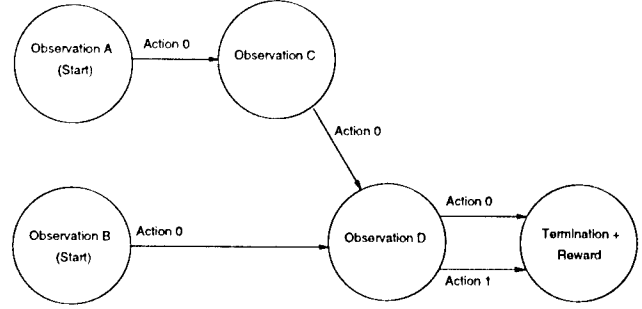


Figure 2: The hPOMDP discussed in the proof of Theorem 2.

action available from intermediate observation C; and two actions available from the penultimate observation D. An action from observation A leads to observation C without reward; actions from observations B and C lead to observation D without reward. Both action 0 and action 1 from observation D immediately lead to termination and a reward; the decision process is non-Markovian because the reward depends not only on the action taken from observation D, but also on the starting observation.

We assume that $\gamma < 1$ for this discounted reward decision process; suppose the reward schedule is as follows:

Start observation	action D	reward
A	0	r_1
A	1	r_2
B	0	r_3
B	1	r_4

Let π_0 and π_1 be the policies that correspond to 0 and 1 being the policy action from D. We set $r_1 \dots r_4$ such that $Q^{\pi}(D, 0) > Q^{\pi}(D, 1)$ for arbitrary π (i.e. $(r_1 + r_3)/2 > (r_2 + r_4)/2$), but also so that $J(\pi_0) < J(\pi_1)$ (i.e. $(\gamma r_1 + r_3)/2 < (\gamma r_2 + r_4)/2$). For example, let $r_2 = 0$, $r_3 = 1$, $r_4 = 2$, and select r_1 such that $\gamma r_1 < 1 < r_1$.

In such a case, D will see action 0 as preferable, which appears locally optimal even though the choice results in suboptimal policy π_0 . Thus the sole optimal policy π_1 does not represent a learning equilibrium for this hPOMDP. \square

Next, we examine the case where TD style returns are used:

Theorem 3 *If a $TD(\lambda)$ credit-assignment method is used for direct RL of a NMDP, then for $\lambda < 1$ it is not guaranteed there exists an optimal observation-based policy representing a learning equilibrium.*

Proof Consider the hPOMDP in Figure 3. Observations A and B are the equiprobable starting observations. We note all the transitions are deterministic, and that in observation A there are two actions to select from while observations

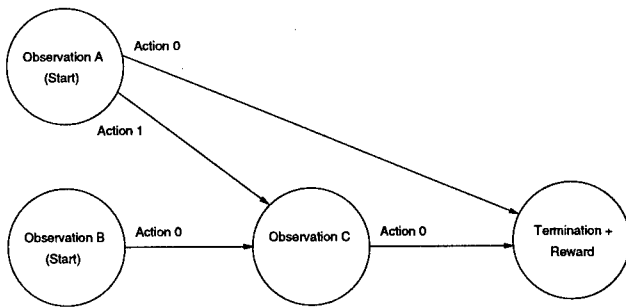


Figure 3: The hPOMDP discussed in the proof of Theorem 3.

B and C have only one. Action 0 from observation C leads directly to termination with an immediate reward; if the starting observation is A, the immediate reward will be zero. Action 0 from observation A also has a termination and a non-zero immediate reward associated with it, the exact value of which we will discuss in a moment. All other transitions have a zero immediate reward associated with them.

The expected value of $\langle C, 0 \rangle$ for an observation based policy π depends upon the relative frequency of the transitions $A \rightarrow C$ and $B \rightarrow C$; this in turn depends upon how often action 1 is selected from observation A for the sake of active exploration. We make no special assumptions regarding an active exploration strategy: we only assume the relative frequencies of action 0 and action 1 selections from observation A are both non-zero; hence $Q^\pi(C, 0) \in (0, 0.5)$.

From the rules of TD updates we can derive that $Q^\pi(A, 1) = \gamma(\lambda \cdot 1 + (1 - \lambda)Q^\pi(C, 0))$, assuming $\gamma \in [0, 1]$. This interests us, because $Q^\pi(A, 1)$ would equal γ under a Monte Carlo method of credit assignment, but for TD(λ) returns $Q^\pi(A, 1) < \gamma$ for all $\lambda < 1$.

Therefore, if the value of the immediate reward for $\langle A, 0 \rangle$ is such that $Q^\pi(A, 1) < Q^\pi(A, 0) < \gamma$, then observation A would see action 0 as preferable to action 1, even though the optimal policy corresponds to selecting action 1. In such a case, the optimal observation-based policy for this hPOMDP does not represent a learning equilibrium if TD(λ) returns are used with $\lambda < 1$. \square

Firstly, we should point out that proof of Theorem 3 has been constructed so that it applies equally to both on-policy methods, such as SARSA (e.g. Singh & Sutton, 1996), and to off-policy methods, such as Q-learning.

Further, if we consider the special case of $\lambda = 0$, we can use this proof to additionally arrive at the following result:

Corollary 1 *If a 1-step Q-learning (or 1-step SARSA) method of credit assignment is used for direct RL of a NMDP, then it is not guaranteed that there exists an optimal observation-based policy representing a learning equilibrium.*

We note we can also use the proof of Theorem 3 to settle a conjecture in (Singh et al., 1994) regarding the optimality of Q-learning for observation-based policies of POMDPs. The authors of that paper conjectured that Q-learning in general might not be able to find the best deterministic memoryless (i.e. observation-based) policy for POMDPs. If we consider $\lambda = 0$ case (i.e., the case corresponding to 1-step Q-learning), this result follows directly from the proof, since the optimal policy for the hPOMDP used in the proof of Theorem 3 is in fact also deterministic.

We also note that in (Pendrith & McGarity, 1997) is a proof that extends these results from 1-step to multi-step corrected truncated returns (CTRs). We omit the proof here for space reasons.

Taken together, these results show that the key property of optimal observation-based policies being stable for direct RL methods does not generalise from Markovian to non-Markovian domains. The stability of optimal observation-based policies under standard RL methods can be guaranteed for hPOMDPs, a general class of NMDPs, only if the additional restrictions of using undiscounted rewards and using actual return credit assignment methods are imposed. These results apply for stochastic as well as for deterministic optimal observation-based policies.

7 Related work

The POMDP theoretical framework was originally formulated in the context of a set of operations research (OR) problems; the wider RL literature reflects an important line of research that is bringing OR methods to bear on the general problem of discovering effective policies in partially observable stochastic domains (Kaelbling et al., 1995). In contrast to “direct” methods of RL for POMDPs, however, these methods generally rely on state-estimation techniques that attempt to disambiguate observations into true Markov states.

Although an analysis of direct RL for POMDPs as presented in this paper might *prima facie* seem to have little bearing on such approaches, this is not necessarily the case. We consider that even if we are using active state-estimation techniques in a POMDP setting, the problem will remain non-Markov to some degree or another while the state-estimation is imperfect; and, in general, the problem of state-disambiguation has been shown to be difficult

(Littman, 1996).

In (Littman, 1994) is a complexity analysis of the general problem of finding the optimal deterministic memoryless (i.e., observation-based) policy for an NMDP. In the general case, this turns out to be NP-complete. More optimistically, in the same paper there is evidence presented that heuristic methods for searching the policy space might be expected to find very good or even optimal policies in the average case.

In (Singh et al., 1994) is proposed a framework for the analysis of direct RL for NMDPs; it is built around a class of POMDPs conceptually similar to hPOMDPs in several important respects.

The authors analyse what two different 1-step TD RL methods (TD(0) and 1-step Q-learning) will learn as value functions for the class of POMDPs under consideration. While they do not continue to a full analysis of TD(λ) for general $\lambda < 1$, they do point out that a Monte Carlo method like TD(1) will result in accurate value estimates for an example POMDP they analyse.

As noted earlier, they conjecture that, in general, 1-step Q-learning is not guaranteed to learn even the best deterministic observation-based policy for a POMDP. However, their analyses are concentrated on the issue of the accuracy of observation-based value estimation, rather than on the stability of optimal policies, which has been our primary focus. Also, as a consequence of the multiple-visit definition of $V(s)$ in their framework, their analysis was necessarily restricted to the asymptotic behaviour of ergodic systems, a limitation which does not apply to the framework presented here.

The analyses of cooperative learning automata in Markov settings by Witten (1977) and Wheeler & Narendra (1986) provided the game theoretic perspective facilitating the original intuitions and reasoning leading to the results presented in this paper.

8 Conclusions and Future Work

An analysis of hPOMDPs has proven to be an aid to understanding the theoretical implications of applying standard discounted reward RL methods to non-Markov environments. Extending earlier work, the framework we present applies to non-ergodic as well as discounted reward NMDPs, facilitating a more direct understanding of the issues involved.

Our analysis starts with the simple observation that having a global maximum in policy space which is also a learning equilibrium is a necessary condition for convergence

to an optimal policy under a given learning method. We discover that for an important general class of non-Markov domains, undiscounted, actual return RL methods have significant theoretical advantages over discounted returns and TD methods of credit-assignment.

A move from discounted to undiscounted rewards naturally suggests a closer look at average reward RL methods for equilibrium properties in non-Markov environments. Some steps in this direction have already been made in (Singh et al., 1994) and (Jaakkola, Singh, & Jordan, 1995). Theorem 2 may point to subtle problems translating "transient reward" sensitive metrics such as Blackwell optimality (Mahadevan, 1996) from MDPs to NMDPs. Investigations are continuing in this direction.

Acknowledgments

The authors wish to thank Pawel Cichosz and Tobias Schefter for their helpful comments on early versions of this paper.

References

- Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72, 81–138.
- Barto, A., Sutton, R., & Anderson, C. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5), 834–846.
- Billingsley, P. (1986). *Probability and measure*. John Wiley & Sons.
- Jaakkola, T., Singh, S., & Jordan, M. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*. Morgan Kaufmann.
- Kaelbling, L., Littman, M., & Cassandra, A. (1995). Planning and acting in partially observable stochastic domains. *Submitted to Artificial Intelligence*.
- Littman, M. (1994). Memoryless policies: theoretical limitations and practical results. In D. Cliff, P. Husbands, J.-A. Meyer, & S. W. Wilson (Eds.), *Proc. of the Third Int. Conf. on the Simulation of Adaptive Behavior*. MIT Press.
- Littman, M. (1996). *Algorithms for sequential decision making*. Ph.D. thesis, Technical report CS-96-09, Dept. of Computer Science, Brown University.

- Mahadevan, S. (1996). Sensitive discount optimality: Unifying discounted and average reward reinforcement learning. In L.Saitta (Ed.), *Machine Learning: Proc. of the Thirteenth Int. Conf.* Morgan Kaufmann.
- Michie, D., & Chambers, R. (1968). BOXES: An experiment in adaptive control.. In E.Dale, & D.Michie (Eds.), *Machine Intelligence 2*, pp. 137–152. Edinburgh: Edinburgh Univ. Press.
- Pendrith, M., & McGarity, M. (1997). An analysis of non-Markov automata games: Implications for reinforcement learning. Tech. rep., UNSW-CSE-TR-9702, School of Computer Science and Engineering, University of NSW, Australia.
- Pendrith, M., & Ryan, M. (1996). Actual return reinforcement learning versus Temporal Differences: Some theoretical and experimental results. In L.Saitta (Ed.), *Machine Learning: Proc. of the Thirteenth Int. Conf.* Morgan Kaufmann.
- Puterman, M. (1994). *Markov decision processes : Discrete stochastic dynamic programming*. New York: John Wiley & Sons.
- Singh, S., Jaakkola, T., & Jordan, M. (1994). Learning without state-estimation in partially observable Markovian decision processes. In W.Cohen, & H.Hirsh (Eds.), *Machine Learning: Proc. of the Eleventh Int. Conf.* New Brunswick, New Jersey: Morgan Kaufmann.
- Singh, S., & Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3), 123–158.
- Watkins, C. (1989). *Learning from Delayed Rewards*. Ph.D. Thesis, King's College, Cambridge.
- Wheeler, Jr., R. M., & Narendra, K. S. (1986). Decentralized learning in finite Markov chains. *IEEE Trans. on Automatic Control*, AC-31(6), 519–526.
- Witten, I. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34, 286–295.

A Randomized ANOVA Procedure for Comparing Performance Curves

Justus H. Piater
piater@cs.umass.edu

Paul R. Cohen
cohen@cs.umass.edu

Xiaoqin Zhang
xqzhang@cs.umass.edu

Michael Atighetchi
adi@cs.umass.edu

Computer Science Department
University of Massachusetts
Amherst, MA 01003

Abstract

Three factors are related in analyses of performance curves such as learning curves: the amount of training, the learning algorithm, and performance. Often we want to know whether the algorithm affects performance and whether the effect of training on performance depends on the algorithm. Analysis of variance would be an ideal technique but for carryover effects, which violate the assumptions of parametric analysis of variance and can produce dramatic increases in Type I errors. We propose a novel, randomized version of the two-way analysis of variance which avoids this problem. In experiments we analyze Type I errors and the power of our technique, using common machine learning datasets.

1 INTRODUCTION

A common task in machine learning is comparative assessment of learning methods. Most research on this issue focuses on performance measures such as classification accuracy after training, or percentage of games won by a game-playing program (e.g. Mitchell 1997 ch. 5, Dietterich (in press), Rasmussen et al. 1996). However, it is sometimes interesting to compare time series of performance, such as learning curves. For example, two algorithms might have comparable asymptotic performance, but we would like to test the hypothesis that one achieves this level of performance more quickly than the other.

Which statistical procedures are appropriate to identify differences between the performance of algorithms over time, and particularly during training? One obvious approach might be to apply the aforementioned methods repeatedly

at different times, comparing the performance of algorithms at each of several levels of training. Unfortunately, multiple comparisons can lead to overestimates of the significance of results (see Section 2) and are inappropriate for comparing performance curves.

A better approach is to describe differences between algorithms during training in terms of two effects:

Algorithm Effect: Does one algorithm generally achieve higher performance than another?

Interaction Effect: Does the influence of training on performance depend on the algorithm?

Figures 1a and 1b illustrate prototypical cases for each effect. In practice, however, some combination of effects will occur. In Figure 1c, for instance, both curves start out with similar slopes, but one of them converges to a lower asymptote. Figure 1d shows a case where both curves start at the same point and achieve similar asymptotic performances, but one algorithm learns faster (with respect to the amount of training) than the other. In this latter case, we find that both algorithm and interaction effects concentrate in the early stages of training, and both effects essentially disappear with increasing amount of training.

This paper presents a method for detecting Algorithm and Interaction effects in learning curves. Actually, the method is not restricted to learning curves, it applies to any kind of performance curves. The method tests two hypotheses:

- The mean performances of two or more algorithms are the same (no Algorithm effect).
- The relationship between training and performance does not depend on Algorithm (no Interaction effect).

Such effects are typically tested with *analysis of variance* (ANOVA). However, the conventional parametric ANOVA is

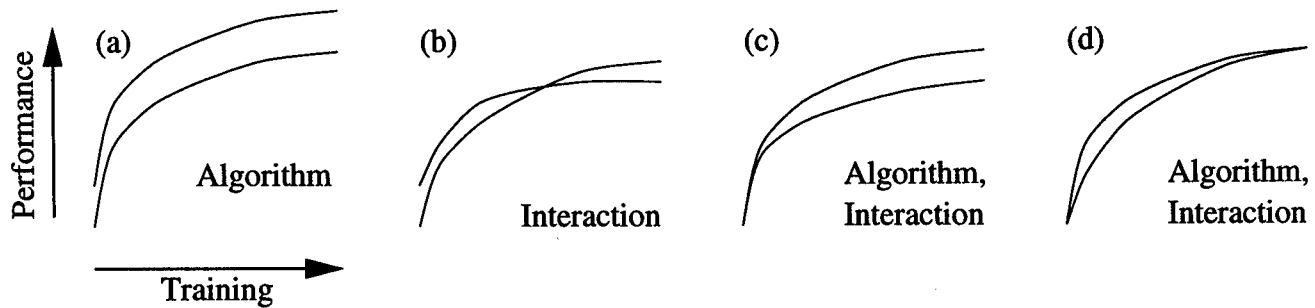


Figure 1: Some kinds of differences between learning curves. The statistical effects on performance (Algorithm and/or Interaction effects) are listed for each situation. In case *c*, the Interaction effect disappears at the later stages of training; in case *d*, both effects disappear.

based on several assumptions, of which one, homogeneity of covariance, is strongly violated by most time series data. In particular, conventional ANOVAs on learning curves can dramatically overestimate the significance of algorithm effects and underestimate the significance of interaction effects. Following some statistical preliminaries in Section 2, we demonstrate how ANOVA gives incorrect results for learning curves (Section 3) and then introduce our novel procedure, a randomized version of ANOVA (Section 4). The remainder of the paper presents experimental results with conventional and randomized ANOVA, comparing the power and Type I errors of the methods.

2 STATISTICAL HYPOTHESIS TESTING

This section defines terms and may safely be skipped by readers familiar with statistical hypothesis testing.

Hypothesis testing involves these steps: Assert a *null hypothesis* H_0 . Decide on a statistic ϕ . Collect a sample s of size n and calculate $\phi(s)$ for the sample. Derive the probability distribution \mathcal{S} of all possible values of $\phi(i)$ for samples i of size n under H_0 . These restrictions are important: \mathcal{S} isn't the distribution of ϕ for *any* sample, but for samples of size n that would arise if the null hypothesis were true. \mathcal{S} is called the *sampling distribution* of ϕ . One may then ask, "What is the probability of obtaining a statistic value of $\phi(s)$ or more by chance if H_0 were true?" The answer, called a *p* value, is the area of \mathcal{S} above $\phi(s)$. Suppose $p = .01$. Should you reject the null hypothesis? There isn't a correct answer to this question, but you can be assured that if you do reject H_0 , the probability that you do so in error is no greater than p . Rejecting H_0 when it is true is called a *Type I error*. Failing to reject H_0 when it is false is a *Type II error*, and the *power* of a test—the probability that you will reject H_0 when it is false—is one minus the probability of a Type II error.

One may also ask, "What value of $\phi(s)$ must I exceed to be assured that my *p* value is less than some threshold α ?" This is called the *critical value* of ϕ and, obviously, it varies with α .

One should not compare performance curves by repeatedly comparing points on the curves (e.g., comparing performance after $i, 2i, 3i \dots$ training instances). Each comparison will with some probability α assert a difference in performance when in reality there is none — a Type I error. If the comparison procedure is applied m times, to m pairs of points on learning curves, then the *total* probability of Type I error is roughly $1 - (1 - \alpha)^m$. (The probability is exactly $1 - (1 - \alpha)^m$ if the comparisons are independent, but they are not, and their non-independence necessitates the technique developed in this paper.) One can control the total probability of a Type I error, but only by reducing α — which increases the critical values for individual comparisons — making it less likely that comparisons will find differences that actually exist. Said differently, the power of the tests is reduced (see Cohen 1995 for a discussion of related issues). Multiple comparisons are not the right tool for comparing performance curves.

3 ANOVA FOR COMPARING PERFORMANCE CURVES

Suppose we have two learning algorithms A_1 and A_2 , each of which trains l times on a set of k instances, e.g., in an l -fold cross validation procedure. Then we have l estimates of the performance of each algorithm at each level of training. Put another way, we have l "lines" $L_1^{(1)}, \dots, L_l^{(1)}$ for A_1 and another l lines $L_1^{(2)}, \dots, L_l^{(2)}$, where each line is a list of k numbers that represent the performance of the algorithm at level h ($1 \leq h \leq k$) of training, on that particular fold of the cross validation. A schematic data table is shown in Figure 2, where the axes of the table represent

the factors *Training* and *Algorithm*. Lines may of course be generated by methods other than cross-validation; for example, they might represent training on several different datasets. The important thing is that the data points on a line are not independent. In statistical parlance, they are *repeated measures* and they create *carryover effects*, meaning that the performance represented by earlier points on a line influences, or carries over to, later performance.

		Training				
		t_1	t_2	t_3	\dots	t_k
A_1		•	•	•		•
		•	•	•		•
		•	•	•		•
		•	•	•		•
A_2		•	•	•		•
		•	•	•		•
		•	•	•		•
		•	•	•		•
\vdots						
A_m		•	•	•		•
		•	•	•		•
		•	•	•		•
		•	•	•		•

Figure 2: Data table setup for randomized ANOVA. This example shows $l = 4$ learning curves per algorithm.

Were it not for these carryover effects, analysis of variance would be an ideal tool to analyze learning curves. Analysis of variance tests for *main effects* of factors and *interaction effects* between factors. Each kind of effect is represented by an F statistic, which has an expected value of 1.0 under the null hypothesis of no effect. Formulae for calculating F are straightforward and widely available (e.g., see Cohen 1995) and will not be repeated here. The patterns of data in Figure 1 can be discriminated by F statistics for main and interaction effects.

Carryover effects make it difficult to specify the sampling distributions of F statistics. Classical F distributions are derived under some assumptions, and while F tests are robust against departures from most of these, learning curves violate an important one: homogeneity of covariance. To see what this means, note that we could calculate a correlation between the four data points in the A_1, t_1 cell of Figure 2 and the four in the A_1, t_2 cell. Under homogeneity of covariance, this correlation would be constant for any pair of cells A_k, t_i and A_k, t_j . However, the correlation between

performance after t and $t + 1$ training instances is apt to be higher than the correlation between performance after t and $t + 100$ instances, so homogeneity of covariance is apt to be violated. The consequence is that the Type I error probabilities no longer correspond to the given α level (Cohen 1995 (p. 306), Keppel 1973, O'Brien and Kaiser 1985).

So F statistics can represent the effects in Figure 1, nicely, but carryover effects bias the p values of the statistics. Can we salvage ANOVA and F tests? One common tactic is to correct statistics to compensate for biases. The following experiment (and those in Sec. 5) shows that this tactic will not work. We generated learning curves from three different datasets (Chess, RL, and Tic-Tac-Toe; see the Appendix). The results (Figure 3) demonstrate a dramatic increase in Type I error in the case of Algorithm effects, and a decrease for Interaction effects. The histograms demonstrate that the frequencies of these errors depend on the dataset, which implies that one cannot correct the F statistics with a simple adjustment. In particular, the Chess and Tic-Tac-Toe learning curves were generated according the same procedure, their degrees of freedom are identical, and yet their mean rejection rates differ dramatically.

Another way to salvage ANOVA is to somehow find the appropriate sampling distributions for F statistics when homogeneity of covariance is violated. This would allow us to control Type I errors precisely. Our method, discussed in Section 4, yields these sampling distributions, and accurate p values, whether or not homogeneity of covariance is violated. The procedure is based on *randomization* (see, e.g., Cohen 1995, ch. 5). Consider first the null hypothesis that *Algorithm* has no effect on performance. If it were true, then the lines associated with algorithm A_1 in Figure 2 might equally well be associated with A_2 , or with any other algorithm. Thus, if we randomly redistribute lines among algorithms, and then calculate F_{alg} in the usual way, we will derive one value of F_{alg} under the null hypothesis that *Algorithm* is independent of performance. For clarity, denote this statistic F_{alg}^* to remind us that it was derived by randomization, that is, shuffling lines, and to distinguish it from the sample statistic F_{alg} that was calculated from the original (unshuffled) data table. If we shuffle the lines again, we will get another, somewhat different value of F_{alg}^* , and if we shuffle 1000 times we can get a distribution of 1000 values of this statistic.

By shuffling lines instead of, say, individual data points among algorithms, we preserve the dependencies among the data points on each line. Said differently, we treat a line as a unit for the purpose of estimating the distribution of F_{alg}^* , so the degree of dependence among the data on a line is irrelevant. As mentioned above, when homogeneity of covariance is violated, comparing F_{alg} to a conventional F

Initialize $c = 0$. Then do 1000 times:

1. Generate a set L of learning curves using C4.5.
2. Partition L randomly into L_1 and L_2 representing two different imaginary algorithms, with $|L_1| = |L_2| = \frac{|L|}{2}$.
3. Perform conventional ANOVA on these data, obtaining the probability p that it is incorrect to reject the null hypothesis that there is no effect of Algorithm on performance.
4. If $p < 0.05$ then increment c .

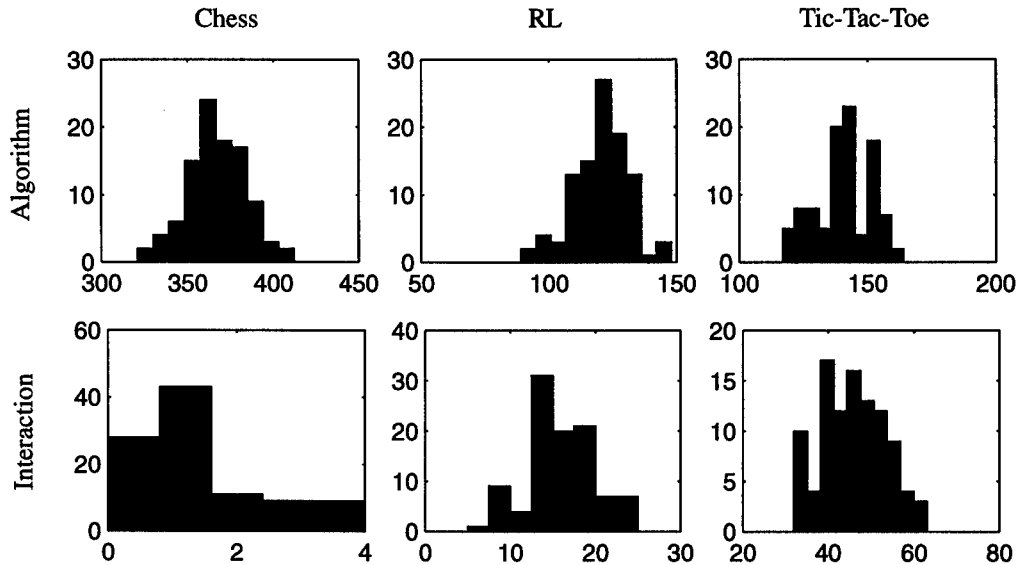


Figure 3: Illustration of the increase in Type I error resulting from carryover effects. For each dataset, the procedure given above was executed 100 times and the resulting c values averaged. Without carryover effects, one would expect $c = 1000\alpha = 50$. The histograms of c values show that H_0 was rejected much more frequently, which demonstrates the inappropriateness of the conventional ANOVA for comparison of learning curves. See the Appendix for details about the datasets used.

distribution will underestimate p , that is, it will make F_{alg} look significant at a given level of α when it is not. The distribution of F_{alg}^* protects against this error, as illustrated by Figure 4.

F_{alg}^* is not technically a sampling distribution but it serves the same purpose, namely, to estimate a p value for a sample result, or to find a critical value that F_{alg} must exceed to reject H_0 with some level α of confidence (Cohen 1995, p. 175).

4 THE PROCEDURE IN DETAIL

Consider a set A of m learning algorithms A_1, \dots, A_m . For each algorithm A_i we have a set $L^{(i)}$ of l learning curves $L_1^{(i)}, \dots, L_l^{(i)}$. Each learning curve $L_j^{(i)}$ constitutes a k -tuple $(L_{j,1}^{(i)}, \dots, L_{j,k}^{(i)})$ of real numbers, where each $L_{j,h}^{(i)}$ gives the performance score of the learning algorithm A_i on the j th run after A_i has performed an amount t_h of train-

ing.¹ Note that k and the t_h ($1 \leq h \leq k$) are the same for all algorithms, but l , the number of learning curves generated by an algorithm, need not be the same for all algorithms.

We will test two null hypotheses: There is no effect of *Algorithm* on performance, and there is no effect of *Algorithm* on the relationship between *Training* and performance. These correspond to F tests of a main effect and the interaction effect in a two-way analysis of variance, so we will compute the appropriate statistics, F_{alg} and F_{int} , but we will compare them to the randomized sampling distributions of F_{alg}^* and F_{int}^* .

The complete procedure can be summarized as follows:

1. For each algorithm i , collect l learning curves $L_1^{(i)}, \dots, L_l^{(i)}$. If there are m algorithms, this will pro-

¹The "amount of training" is an abstract notion here which could be given by the number of training instances processed, the number of trials run, or even by the training time.

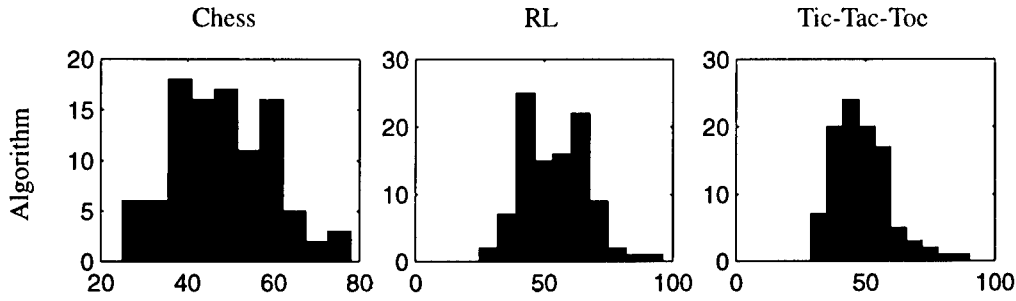


Figure 4: Histograms generated by the same procedure as Figure 3, but p -values were compared against randomized F distributions (500 shuffles) instead of the parametric distributions. In fact, the mean rejection rates of around 50 correspond to the target significance level of $\alpha = 0.05$. This is also true for the corresponding histograms for the Interaction effect (not shown).

duce a data table like the one in Figure 2.

2. Run a conventional two-way analysis of variance on this data table to obtain sample statistics F_{alg} and F_{int} .
3. Generate the sampling distributions F_{alg}^* and F_{int}^* :

Throw the $m \times l$ learning curves into a “pool” \mathcal{P} .

Do $i = 1 \dots z$ times (where z is large, e.g., 1000):

- (a) Shuffle \mathcal{P} and reassign each of the ml learning curves to the m algorithm categories (rows in the data table) such that each row contains l curves. Shuffling \mathcal{P} enforces the null hypothesis of no association between performance and algorithm.
- (b) Run a conventional two-way analysis of variance on the resulting data table and record $F_{\text{alg},i}^*$ and $F_{\text{int},i}^*$.

4. Find the critical values in the distributions F_{alg}^* and F_{int}^* . If $\alpha = .05$ and $z = 1000$ then the critical value in each sorted distribution is the 950th, because 5% of the distribution lies above this value. In general, the critical value is the $\alpha 1000$ th quantile.
5. If F_{alg} exceeds the critical value for the F_{alg}^* distribution, reject the null hypothesis that *Algorithm* does not affect performance. Similarly if F_{int} exceeds the critical value for the F_{int}^* distribution, reject the null hypothesis of no interaction effect.
6. The p value for each hypothesis is derived from the rank of the closest value in the sorted sampling distribution. For example, if $F_{\text{alg}} = 10.3$ and the closest value in F_{alg}^* is 10.2, and if the rank of this value is 972 out of 1000, then $p < (1000 - 972)/1000 = .028$.

5 EXPERIMENTAL RESULTS

In Section 3 we illustrated the increase in Type I error caused by comparing F statistics to standard F distributions. This section provides a more detailed account of this phenomenon. Both Algorithm and Interaction effects are analyzed on the Chess dataset (see Appendix). The following section discusses the probability of Type I error, and Section 5.2 compares the power of the conventional and randomized ANOVAs. In all cases we use $m = 2$ sets of learning curves. Note that our method applies to any $m \geq 2$.

5.1 TYPE I ERROR MEASUREMENTS

As shown in Section 3, the standard F distributions tend to overestimate the significance of Algorithm effects, but underestimate the Interaction effects. We expected the overestimations based on previously published results (e.g., Keppel 1973, p. 464) but the underestimations were a surprise and we do not have a satisfactory explanation for this phenomenon. In one sense, we do not care why the standard F distributions detect Interaction effects less often than expected, because we have a method to construct correct F distributions. Yet we were curious. To shed some light on this issue, we examined the frequency of Type I errors for Interaction and Algorithm effects, for conventional ANOVA and our method, in a variety of conditions.

Recall that Type I error rates are the frequencies with which the null hypothesis is rejected when it is true, i.e., when there is no effect. In Section 3 we enforced the null hypothesis by splitting a set of learning curves generated by one algorithm into two groups, calling one group “algorithm A,” the other “algorithm B,” then testing for an Algorithm effect and an Interaction effect. Because the two groups were generated by one algorithm, we expected neither effect; that is, we expected Type I error rates of α . In

the following experiments we enforce the null hypothesis in a slightly different way. First we generated a set L of learning curves with C4.5, then to each curve we applied a transformation, yielding another set L' . The transformation induced an Algorithm effect or an Interaction effect or both. In other words, the mean curves for L and L' correspond to the pairs of curves in Figure 1. Then, to enforce the null hypothesis, we shuffled the curves in L and L' . Whereas the earlier procedure enforced the null hypothesis by randomly dividing a set of statistically-identical learning curves, this procedure is more natural in starting with two sets of curves (L and L') that are different, then shuffling them. Moreover, we have tight control over the degree of difference between L and L' because we transform the former to get the latter.

We now describe this procedure in detail. The following steps compute the number c of rejections of H_0 during 1000 analyses of variance, starting from a set L of learning curves:

Initialize $c_{\text{conv}} = c_{\text{rand}} = 0$. Then do 1000 times:

1. Construct L' by modifying each curve from L according to one of the cases given in Figure 1. The degree of modification is controlled by a factor f . We will denote this operation by $L' = M_a(L, f)$ for case a in Figure 1, and likewise for cases b, c, d .
2. Partition $L \cup L'$ randomly into L_1 and L_2 , with $|L_1| = |L_2| = 20$.
3. Perform conventional ANOVA on these data to obtain the F statistic for the tested effect.
4. Compare F to the appropriate conventional F distribution and read off the probability p_{conv} that it is incorrect to reject H_0 .
5. Generate a randomized sampling distribution F^* using 400 shuffles as described in Section 4 item 3, and read off p_{rand} .
6. If $p_{\text{conv}} < \alpha$ then increment c_{conv} .
If $p_{\text{rand}} < \alpha$ then increment c_{rand} .

This procedure was performed with respect to Algorithm and Interaction effects, and for 10 different values of f . For each of these cases, the c values resulting from 10 such runs were averaged to yield a data point shown in Figure 5. The effect of the modification factor f on the shape of a curve is also illustrated in the figure. Details on the four modification procedures are given in the Appendix.

As expected, the randomized ANOVA always achieves Type I error probabilities near the target significance level

of $\alpha = 0.05$. The conventional method, however, tends to assert an Algorithm effect too often (increase in Type I error probability). In contrast, Interaction effects are mostly detected less often than the expected 5%.

Modification M_b is a dramatic case: This modification did not introduce an Algorithm effect, and yet such an effect was often detected by the conventional ANOVA at a frequency inversely proportional to the modification factor f . The modification introduced an Interaction effect which was then shuffled away, enforcing the null hypothesis of no interaction, yet the frequency with which conventional ANOVA detected Interaction effects increases with f . We do not know why, and these experiments fail to explain why Type I errors for interaction effects are lower than expected, although the dependence on f is intriguing.

The magnitude of these misjudgments can be quite dramatic (up to a factor of ten in these examples), but depends on the type of the effect and the modification factor f . Because of these dependencies, we think it is not possible to correct the standard F statistics to control Type I errors precisely. No matter: Our randomized ANOVA produces the expected Type I errors.

5.2 POWER MEASUREMENTS

Whereas Type I errors involve detecting effects that don't exist, Type II errors involve failing to detect errors that do exist. The *power* of a test is one minus the Type II error rate, that is, the probability of detecting a true effect. To measure the power of both conventional and randomized versions of ANOVA, we employed the same modification strategy as in the previous section. Here, however, L and L' are not shuffled. In other words, L and L' give us controlled Algorithm and Interaction effects. The following procedure measures the power of both ANOVAs to detect these effects:

1. Construct $L_2 = M_x(L_1, f)$, where x is one of a, \dots, d .
2. Generate a randomized sampling distribution F^* , as described in Section 4 item 3, using 500 shuffles of 2×10 learning curves each.
3. $c_{\text{conv}} = c_{\text{rand}} = 0$.
4. Do 100 times:
 - (a) Randomly draw a set L'_1 of 10 unique curves from L_1 .
Randomly draw a set L'_2 of 10 unique curves from L_2 .
 - (b) Perform conventional ANOVA and obtain F .

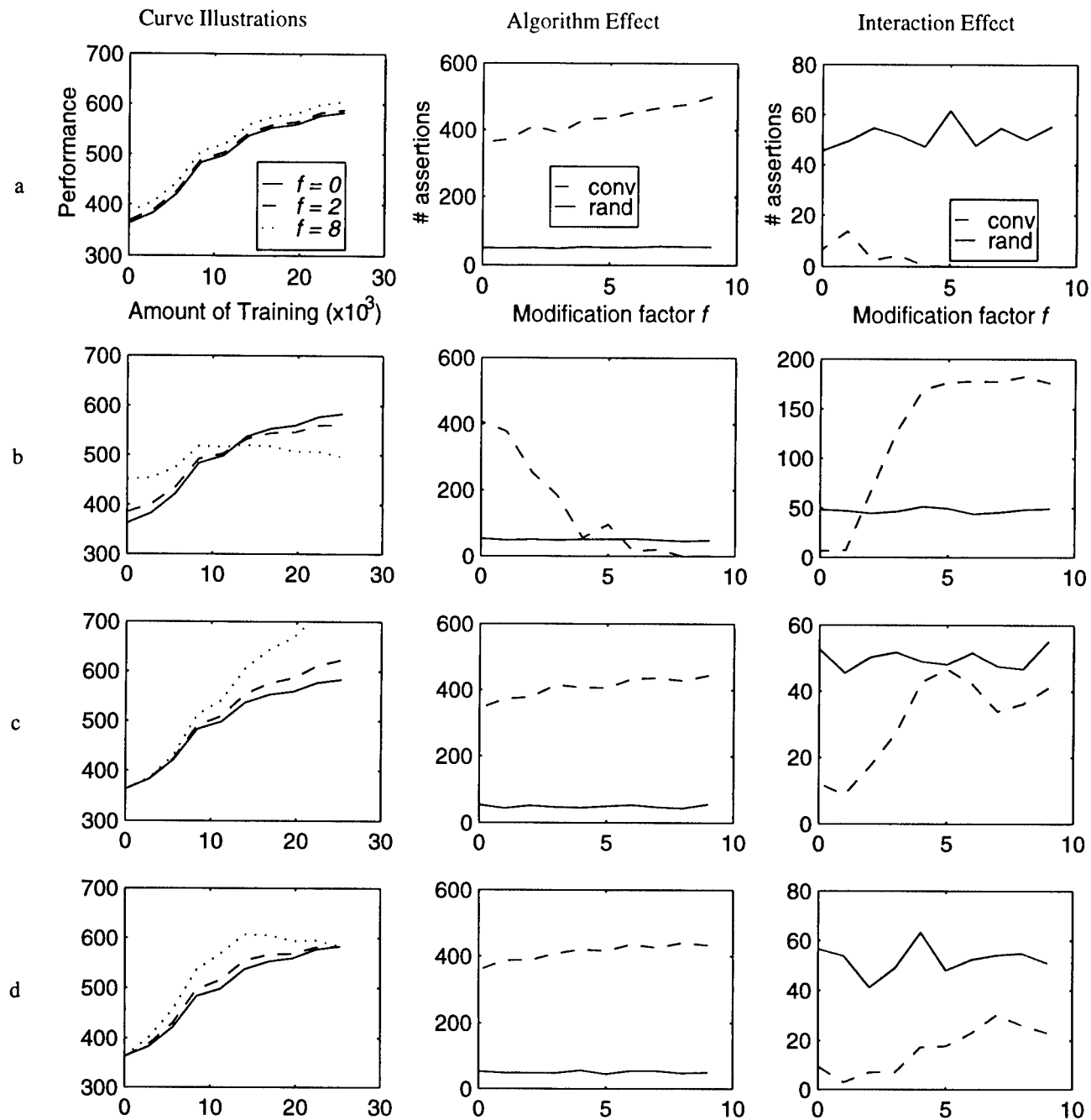


Figure 5: Effects asserted by the conventional and randomized ANOVA methods. Each row shows one of the modification cases *a–d* from Figure 1. The left column illustrates the effect of the modification for different values of f ($f = 0$ means no modification). The center and right columns plot the number of times (of 1000) the conventional and randomized analyses asserted an Algorithm or Interaction effect at $\alpha = 0.05$.

- (c) Compare F to the parametric F distribution and obtain p_{conv} .
Compare F to the randomized F^* distribution and obtain p_{rand} .
- (d) If $p_{\text{conv}} < \alpha$ then increment c_{conv} .
If $p_{\text{rand}} < \alpha$ then increment c_{rand} .

Divide c_{conv} and c_{rand} by 100 to obtain the power measurements.

This procedure was performed to introduce Algorithm and Interaction effects for 10 different values of f . For each of these cases, the c values resulting from 8 such runs were averaged to yield a data point shown in Figure 6.

As in earlier experiments, the conventional ANOVA usually overestimates the presence of an Algorithm effect, thus it appears more powerful than our randomized ANOVA. But this "power" is illusory, like a watchdog that barks all night whether or not a prowler is on the premises. Sure, the dog will bark when there is a prowler — the probability of detecting a prowler is 1.0 — but it is a useless animal. In modifications a , c and d , where Algorithm effects are present, our method detects them handily and at a Type I error rate of approximately 5%. In case b , where there is no algorithm effect, our method does not report one, but the conventional method does. Similarly, for interaction effects, our method does not detect one in case a , because none exists, and it is quite powerful in the other cases, where interaction effects are present.

6 CONCLUSION

We have presented a statistical method for comparing sets of performance curves, such as learning curves, when points on the curves are not independent, that is, when there are carryover effects and homogeneity of covariance is violated. We demonstrated that in these conditions conventional analysis of variance produces a sometimes dramatic surplus of Type I errors for main (algorithm) effects and a shortfall of Type I errors for interaction effects. Because the magnitude of these surpluses and shortfalls depends on the original dataset, among other things, we do not think they can be corrected by adjusting conventional F statistics. Instead we show how to construct sampling distributions for the F statistics that correct for violations of homogeneity of covariance. With this method, one can control error rates precisely. We recommend the method for its simplicity and hope it will be a helpful addition to the statistical toolbox of the machine learning community.

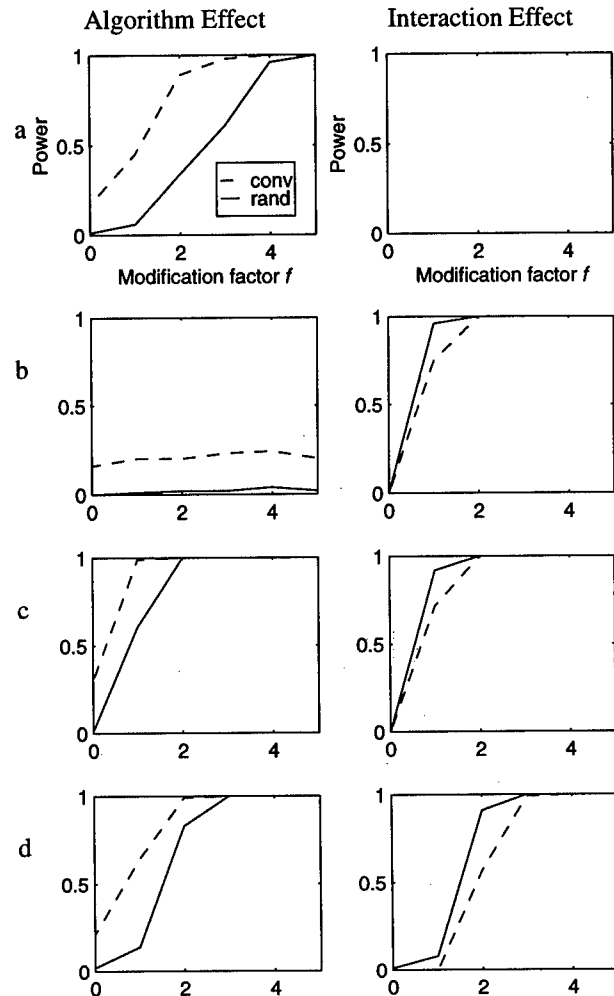


Figure 6: Power measurements of the conventional and randomized ANOVA methods. Each row shows one of the modification cases a – d from Figure 1. The horizontal axes indicate the degree f to which one of one underlying two sets of curves was modified with respect to the other (see Figure 5).

Appendix: Sources of Learning Curves

Chess: Chess Endgame Database (king-rook-vs-king, Bain 1994) provided by the UCI Machine Learning Repository (Merz and Murphy 1996). Twenty Learning curves were generated by running the decision tree algorithm C4.5 (Quinlan 1993) in a 20-fold cross validation procedure.

We now describe the modification functions $M_x(L, f)$ used in Section 5. In the following, r refers to the difference between the performance values of the last and first points of a given learning curve, i.e. $r = L_k - L_1$. For each learning curve L , each performance value L_i is altered according to a given modification case (cf. Figure 1):

- (a) $L_i = L_i + f \frac{r}{80}$
- (b) $L_i = \begin{cases} L_i + f \frac{r}{100} (\frac{k}{2} - i + 1) & \text{if } i \leq \frac{k}{2} \\ L_i - f \frac{r}{100} (i - \frac{k}{2}) & \text{if } i > \frac{k}{2} \end{cases}$
- (c) $L_i = L_i + f \frac{L_i - L_1}{100} (i - 1)$
- (d) $L_i = L_i + \begin{cases} f r \frac{i-1}{100} & \text{if } i \leq \frac{k}{2} \\ f r \frac{k-i}{100} & \text{if } i > \frac{k}{2} \end{cases}$

RL: These data were generated by an AI program that employed TD(0) Reinforcement Learning (Sutton 1988) to learn to play Tic-Tac-Toe against a random opponent. The performance score was the cumulative score of one hundred test games against a random player, where losses, draws and wins scored -1, 0, and 1 respectively. Ten learning curves were generated by one training session each.

Tic-Tac-Toe: Tic-Tac-Toe Endgame Database (Aha 1991) provided by the UCI Machine Learning Repository. Learning curves were generated as with the Chess dataset.

References

- Aha, D. W. (1991). Incremental constructive induction: An instance-based approach. In *Proc. 8th Int. Workshop on Machine Learning*, Evanston, IL, pp. 117–121. Morgan Kaufmann.
- Bain, M. (1994). *Learning Logical Exceptions in Chess*. Ph. D. thesis, University of Strathclyde.
- Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. Cambridge, Massachusetts: MIT Press.
- Dietterich, T. G. (in press). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*.
- Keppel, G. (1973). *Design and Analysis: A Researcher's Handbook*. Englewood Cliffs: Prentice-Hall.
- Merz, C. and P. Murphy (1996). UCI Repository of machine learning databases. <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
- O'Brien, R. G. and M. K. Kaiser (1985). MANOVA method for analyzing repeated measures designs: An extensive primer. *Psychological Bulletin* 97(2), 316–333.
- Quinlan, J. R. (1993). *Programs for machine learning*. Morgan Kaufmann.
- Rasmussen, C. E., R. M. Neal, G. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani (1996). *The DELVE Manual*. University of Toronto, Dept. of Computer Science. <http://www.cs.utoronto.ca/~delve>.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning* 3, 9–44.

Classification Using Φ -Machines and Constructive Function Approximation

Doina Precup

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
dprecup@cs.umass.edu

Paul E. Utgoff

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
utgoff@cs.umass.edu

Abstract

The classification algorithm CLEF combines a version of a linear machine known as a Φ -machine with a non-linear function approximator that constructs its own features. The algorithm finds non-linear decision boundaries by constructing features that are needed to learn the necessary discriminant functions. The CLEF algorithm is proven to separate all consistently labelled training instances, even when they are not linearly separable in the input variables. The algorithm is illustrated on a variety of tasks, showing an improvement over C4.5, a state-of-art decision tree learning algorithm.

1 Introduction

The task of classification is to find an approximate definition for an unknown function $f : \mathbf{X} \rightarrow \{c_1, \dots, c_R\}$, $R \geq 2$ based on a set of training examples of the form $\langle \mathbf{x}_i, f(\mathbf{x}_i) \rangle$. The components of an instance vector \mathbf{x}_i can take values from discrete or continuous domains. It is also possible that the values of one or more components are missing or imprecisely recorded for certain training instances, or that an instance is mislabeled.

This paper presents a different approach to classification, centered around the idea of constructing a machine that is linear in its parameters, but non-linear in the input variables. Therefore, the algorithm constructs a non-linear fit of the data. Unlike decision tree induction, the method does not partition the data into subproblems. The whole training set is used at all the stages of the classifier's construction. The algorithm does not need multiple runs to achieve good results, and finds a perfect separation of the training instances into classes, if one exists. The features it extracts

from the data have a logical form, and thus are easy to interpret.

2 Linear Machines

One approach that constructs a classifier using all the training data is to use linear machines (Nilsson, 1965; Duda & Hart, 1973). A linear machine is a set of R linear discriminant functions g_i used collectively to assign an instance to one of R classes. Let $\mathbf{x} = (1, x_1, \dots, x_n)$ be an instance description. Each discriminant function $g_i(\mathbf{x})$ has the form $\mathbf{w}_i^T \mathbf{x}$, where \mathbf{w} is an $(n+1)$ -dimensional vector of coefficients (weights). An instance is assigned class i if and only if $g_i(\mathbf{x}) > g_j(\mathbf{x}) \forall j \neq i$. If a tie occurs, the instance is attributed randomly to one of the classes.

The training algorithm of a linear machine adjusts its weights based on a set of training instances. The machine starts with arbitrary initial weights, and sweeps through the set of training instances repeatedly. If an instance having class i is erroneously placed into class j , the weight vectors corresponding to the two classes are adjusted as follows: $\mathbf{w}_i \leftarrow \mathbf{w}_i + c\mathbf{x}$ and $\mathbf{w}_j \leftarrow \mathbf{w}_j - c\mathbf{x}$. The amount of correction c can be computed using the fractional error correction rule:

$$c = \alpha \frac{(\mathbf{w}_i - \mathbf{w}_j)^T \mathbf{x}}{2\mathbf{x}^T \mathbf{x}} + \epsilon,$$

where $\alpha \in (0, 1)$ is the step size, controlling the magnitude of the correction, and $\epsilon \geq 0$ controls the "safety margin" between the two classes. If the training instances are linearly separable, this update rule guarantees that the linear machine will converge to a boundary that classifies them correctly.

For many tasks, linear combinations of the input values are not enough to discriminate the groups of instances belonging to each class. When a non-linear discriminant is needed, one possible solution is to use a Φ -machine (Nils-

son, 1965), which is much like a linear machine, except that it uses discriminant functions of the form $g_i(\mathbf{x}) = \mathbf{w}_i^T F_i(\mathbf{x})$, where $F_i = \langle f_1, \dots, f_M \rangle$ is a vector of linearly independent, real, single-valued functions $f_j : \mathbf{X} \rightarrow \mathbb{R}$, independent of the weights. This means that f_j are not varying with the weight adjustments. Multilayered neural networks, for instance, do not satisfy this requirement, since their hidden units change with the weight adjustments.

Φ -machines preserve the theoretical advantages of linear machines, while allowing for non-linear combinations of the inputs. Therefore, Φ -machines can represent partitions of the input space that cannot be represented by linear machines. The training procedures used for linear machines can be applied to adjust the weights of Φ -machines. All the convergence theorems for linear machines apply to Φ -machines as well.

Due to the great variety of classification tasks, one cannot know a priori what mappings f_j would be useful as components of discriminants. It would be useful to construct such functions f_j automatically, based on the training instances.

3 Constructing a Φ -machine for classification

Any method for automatically constructing Φ -machines needs to generate functions f_j that are linearly independent and do not vary when the parameters w_j of the machine are adjusted. Constructive methods that adjust the function while correcting the output weights (by adjusting input weights, for instance) are not suitable candidates, because they generate functions f_j that are not independent of the machine parameters w_j .

In the case of Boolean input variables, one alternative would be to choose f_j from a set of basis functions, such as Rademacher-Walsh or Bahadur-Lazarsfeld polynomials (Duda and Hart, 1973). However, if the f_j are orthogonal (i.e. $f_i \cdot f_j = 0, \forall i \neq j$ and $f_i \cdot f_i \neq 0$), the information that can be gathered during training can only say whether more terms are needed, but not what those terms should be. The search for a good set of discriminant functions is therefore quite difficult.

An automatic method for constructing a Φ -machine adequate for the task at hand is needed. To this end, we use the ELF function approximation algorithm, (Utgoff & Precup, 1998) which constructs new features as needed, by identifying subsets of instances that share intrinsic properties. One could substitute ELF with any other algorithm that can automatically construct linearly independent features.

ELF assumes that the instances are represented using

Boolean input variables. Its goal is to find set covers over the instance space, grouping those instances into subsets that share an intrinsic property, i.e. that can be associated with a common value. Let \mathbf{X} be the space of all describable input instances. An ELF feature is a membership function for a subset of instances $\mathbf{X}_j \subseteq \mathbf{X}$:

$$f_j(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathbf{X}_j \\ 0 & \text{otherwise} \end{cases}$$

When a feature f_j is multiplied by its single corresponding weight, each term $w_j f_j$ has value w_j for the instances that \mathbf{X}_j covers, and 0 elsewhere, thus associating a particular value with a particular set of instances.

The subset \mathbf{X}_j is represented by a pattern vector with as many components as the dimensionality of an instance vector \mathbf{x} . Each component of a pattern has either the value '#' or the value '0'. A '#' matches either of the possible values of the corresponding input vector, while a '0' in the pattern matches only a '0' value. For example, the pattern '#0' covers the instances '10' and '00' and does not cover either '01' or '11'. The pattern of all '#' covers every domain element because the pattern matches any domain element at every component. One pattern is more general than another if and only if it covers all the instances covered by the other, and some additional instances as well.

Initially, each discriminant function consists of one feature, which covers the whole instance space, and has a weight of 0. To evaluate an instance using a discriminant function, one computes the linear combination of the feature values and feature weights. To update the approximation, the training procedure revisits the training instances and adjusts the weights of the discriminant functions using the fractional error correction rule (Nilsson, 1965). Only features that matched the instance have their weights adjusted, because features that did not match have value 0.

For each feature, the algorithm keeps track of the errors associated with each input bit, in order to determine which feature is having the greatest difficulty in fitting. When an adjustment of the weights has ceased to be productive, the algorithm adds a new feature, which is a specialization of the feature that has been producing the largest errors. Specialization is performed by copying the feature and changing a '#' in its pattern to a '0'. The choice of the bit to specialize is based on the variance of the input errors for each feature. The bit whose errors are most different from the mean bit error of the feature is specialized. The new feature will cover half of the set covered by its "parent".

The features that are created by this procedure are linearly independent. The proof of this statement can be done by induction on the number n of bits that are present in an input

instance. Consider the base case, in which $n = 1$. The instance space contains two instances: '0' and '1'. There are two features that can be defined over this instance space: the most specialized feature, which is associated with the pattern '0' and only covers the first instance, and the most general feature, which corresponds to the pattern '#' and covers both instances. The values of the features for each instance can be tabulated in the following determinant:

$$\begin{array}{c|cc} & 0 & \# \\ \hline 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}$$

which can be reduced to a unit determinant, by subtracting the last line from the first one.

Now comes the induction step. Consider the space of the instances that can be generated by n input bits. These instances can be viewed as being generated from the $(n-1)$ -bit instances, by adding a '0' or a '1' on the first position of the vector. Similarly, the features that can be defined over these instances are generated from $(n-1)$ -bit features by adding a '0' or a '#' on the first position of the feature. Let d_{n-1} define the determinant of the $(n-1)$ -bit space input features. The determinant d_n on the n -bit space can be written as:

$$\begin{array}{c|cc} & 0F_{n-1} & \#F_{n-1} \\ \hline 0X_{n-1} & d_{n-1} & d_{n-1} \\ 1X_{n-1} & 0 & d_{n-1} \end{array}$$

The induction hypothesis is that d_{n-1} can be reduced to a unit determinant. This can be done by adding and subtracting lines from each other, as we did in the base case. If there is a sequence of transformations that achieves this goal, we can apply it in the upper and lower part of d_n . The resulting determinant will have the form:

$$\begin{array}{c|ccc|ccc} 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 1 \\ \hline 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 \end{array}$$

By subtracting the bottom half of the determinant from the upper half, d_n can also be reduced to a unit determinant. Thus, the set of all possible features is linearly independent. This means that any subset of features will be linearly independent as well. ■

The process of training CLEF's classifier can be viewed as constructing a sequence of Φ -machines. The previous proof ensures that at any point between two feature additions, the classifier that is built is a Φ -machine. A machine will converge to a set of weights that separates the training instances, if a separation is possible given the current set of features. If no linear separation can be found given the current feature set, by gradually reducing the size of the corrections, the weights will still settle into a particular range (Frean, 1990).

In this case, a new feature will be added, and training will resume with a new machine. In the worst case, the process will continue until all the 2^n features that are possible have been generated. If the instances are separable when mapped through a subset of the features, they will also be separable when the whole set is used. Thus, if a linear separation of the training instances is possible, the algorithm is guaranteed to find one. In practice, CLEF also proved to be quite efficient with respect to the number of features it generates for a particular instance space.

4 Input representation

The non-linear machine described so far requires boolean input values. Such an encoding can be generated automatically for classification tasks. Symbolic variables are mapped into a 1-of- m encoding, where m is the number of possible values for each variable. A variable v with possible values v_1, \dots, v_m is represented in m bits. Bit j will have the value 1 in an instance representation if and only if the test $v(\mathbf{x}) = v_j$ is true.

Since ELF only deals with Boolean inputs, some form of discretization is needed for continuous variables. We have experimented with two methods for discretizing the continuous variables. The first method was suggested by Fayyad and Irani (1993). The basic mechanism is to sort the instance class labels based on the value of the continuous variable. The points at which the class label changes are potential cutpoints for the variable. At each step, the algorithm looks at the list of possible cutpoints and determines the information gain for each partition generated by the cutpoint. A cutpoint is accepted if its information gain is above a certain threshold, and in this case the algorithm proceeds recursively to partition the sub-intervals left and right of the cutpoint. We found this method to be quite conservative in the number of intervals used in the discretization, which led to poor performance when used for our classification algorithm.

The second method was originally proposed by Fulton, Kasif and Salzberg (1995) and then extended by Elomaa and Rousou (1996). In this case, the algorithm searches

for the best split with a given maximum number of intervals. The quality of a partition is evaluated by an impurity measure, and the efficiency of the search is ensured by a dynamic programming algorithm. The impurity measure used for the experiments reported in this paper is information gain. Based on the intervals determined in this way, the continuous values for all the instances are transformed into a 1-of- m encoding, with one bit for each of the m intervals.

The number of bits representing each input variable varies widely. If the input variables were coded in the same number of bits, the probability of any input bit having the value 1 is equal, assuming that all the input instances are equiprobable. For variables coded with different numbers of bits, the probability of a bit corresponding to a low arity variable being on is higher than the probability of a bit being on for a high arity variable. A simple adjustment is used to remove this bias: the error attributed to each bit is normalized with respect to the number of bits used to encode the variable to which the bit belongs.

To handle missing values, if the value of a variable is missing in the input then all the bits corresponding to that variable are set to 0. This prevents the missing value from having any role in the classification process, since it will not interfere with the matching (all features will match at that input variable).

5 Illustration

The Boolean encoding of the features allows an interpretation of the units that form a non-linear classification machine. Feature interpretation can be generated automatically, by printing the negation of each test for which there is a '0' in the feature's pattern.

Table 1 illustrates the features that have been constructed for one of the units (discriminant functions) in the hepatitis task from the UCI data repository (Murphy and Aha, 1994). This is a two-class problem, thus the corresponding linear machine will have two discriminant functions, one for each class. However, due to the training procedure, these discriminant functions are always trained with equal amounts of error having opposite signs. In this two class case, the functions end up having the same features, with weights of opposite sign.

This table is analogous to a "health test", which tells how to compute a score for an instance. For each line in the table, one would check if the instance satisfies the test in the right column. If so, the corresponding weight would be added to the total score. If the total score is positive, the instance would be considered as belonging to the "die" class. For example, a patient with

Table 1: Unit corresponding to the "die" class in the hepatitis task

Hepatitis	
Weight	Feature
-0.019	age $\nless 37.50$
0.013	ascites \neq no
-0.012	age $\nless 37.50$, liver-firm \neq yes, spiders \neq no, varices \neq no
0.008	intercept term
0.008	age $\nless 37.50$, protime $\nless 44.50$
0.008	age $\nless 37.50$, varices \neq no
0.007	age $\nless 37.50$, spiders \neq no, varices \neq no
-0.006	sgot $\nless 80.50$, protime $\nless 87.50$
-0.005	steroid \neq yes
-0.004	bilirubin $\nless 1.35$
-0.004	protime $\nless 87.50$
-0.004	sex \neq female
0.003	sex \neq female, anorexia \neq yes
-0.002	sex \neq female, liver-firm \neq no
0.001	spiders \neq no, histology \neq yes
-0.000	spiders \neq no

the following characteristics: age=30, ascites=yes, spiders=no, sex=female, steroid=no, sgot=79.6, steroid=no, bilirubin=2, protime=80 will be evaluated to a score of $0.013 + 0.008 - 0.005 - 0.004 = 0.0012$, and will therefore be classified as belonging to the "die" class.

6 Analysis

How does CLEF perform compared with other classification algorithms? Will it find a separating Φ -machine in a reasonable amount of time? Will it construct a large number of features, perhaps producing an incomprehensible classifier?

In order to answer these questions empirically, CLEF and C4.5 were run on several classification tasks, mostly from the UCI data repository (Murphy and Aha, 1994). This allows for a comparison in terms of classification accuracy, and provides some insight on the efficiency of CLEF and the form of the function it provides.

The salient difference between CLEF and decision tree inducers is that CLEF uses all the training set to construct its classifier. It should be advantageous to CLEF that it solves one classification problem using all the data, instead of many subproblems, each using only some of the data.

CLEF was trained by repeatedly sampling at random $N = 100|X|$ times from the training set (where $|X|$ is the size of the training set), for a fixed number of epochs. Training can stop early, if the instances in the training set are perfectly separated. For C4.5, the default settings were used

Table 2: Accuracy results

Task	C4.5	C4.5p	CLEF
audio-no-id	75.7 \pm 9.6	77.8 \pm 6.6	79.1 \pm 9.1
balance-scale	78.3 \pm 4.1	77.5 \pm 3.2	92.5 \pm 4.0
breast-cancer	66.2 \pm 6.9	75.5 \pm 3.9	70.3 \pm 7.1
bupa	64.6 \pm 5.3	64.6 \pm 5.6	68.7 \pm 5.0
cleveland	46.8 \pm 4.1	46.8 \pm 5.4	48.7 \pm 8.4
hepatitis	76.9 \pm 4.9	77.5 \pm 5.7	81.9 \pm 5.2
iris	94.4 \pm 7.6	94.4 \pm 7.6	94.4 \pm 7.1
led24	61.0 \pm 9.0	62.4 \pm 9.4	61.9 \pm 11.1
lymphography	77.3 \pm 12.4	78.0 \pm 11.9	80.7 \pm 6.3
monks-2	44.5 \pm 9.3	65.9 \pm 0.0	92.3 \pm 4.8
mplex-6	57.1 \pm 20.2	57.1 \pm 19.2	91.4 \pm 14.6
promoter	80.9 \pm 14.3	77.3 \pm 14.2	87.3 \pm 6.0
soybean	90.3 \pm 2.8	92.2 \pm 2.4	91.9 \pm 3.1
switzerland	32.3 \pm 9.6	33.1 \pm 7.7	35.4 \pm 14.7
tictactoe	66.3 \pm 2.0	68.1 \pm 2.3	78.4 \pm 2.8
va	28.1 \pm 12.7	26.7 \pm 10.0	32.9 \pm 7.2
votes	95.7 \pm 3.7	96.6 \pm 3.3	94.3 \pm 3.1
waveform	69.7 \pm 10.4	70.0 \pm 10.7	73.9 \pm 9.1
wine	93.3 \pm 6.0	93.3 \pm 6.0	94.2 \pm 8.3
zoo	92.7 \pm 6.8	91.8 \pm 6.4	96.4 \pm 4.5
	69.6	71.4	77.3

Table 3: Duncan Multiple Range Test

C4.5	C4.5p	CLEF
69.6	71.4	77.3

(Quinlan, 1993), both with and without pruning. The reason for including the results without pruning as well is that CLEF does not currently use any mechanism for avoiding overfitting. Therefore, using C4.5 without pruning offers some insight into the comparative quality of the learning algorithm itself, though we would like to devise a pruning mechanism for CLEF.

Table 2 shows the accuracy results of the two algorithms, in terms of the mean and standard deviation for each task. All values are computed from a ten-fold stratified cross-validation, with CLEF and C4.5 using the same partitions for each task. As shown in the table, CLEF constructs more accurate classifiers than C4.5 without pruning on 19 of the 20 tasks considered. The classifiers are also more accurate than those constructed by C4.5 with pruning on 15 out of the 20 datasets considered. By doing one-way ANOVA, the difference between CLEF and C4.5 with no pruning is significant at the 0.05 level. The difference with C4.5 with pruning is not statistically significant. These results are confirmed also by the Duncan Multiple Range Test (as shown in Table 3). There is a statistical difference between CLEF and C4.5 without pruning, but there is no statistical difference between CLEF and C4.5 with pruning.

Table 4: Characteristics of the classifier produced

Task	CPU CLEF	Size CLEF	Match
audio-no-id	218.2 \pm 42.2	88.0 \pm 2.8	77.3 \pm 0.8
balance-scale	59.9 \pm 37.8	39.0 \pm 2.3	66.6 \pm 1.9
breast-cancer	191.2 \pm 8.7	47.1 \pm 1.8	47.7 \pm 1.6
bupa	245.4 \pm 35.8	49.2 \pm 2.4	64.9 \pm 4.6
cleveland	496.0 \pm 46.2	117.4 \pm 6.1	72.1 \pm 2.6
hepatitis	58.8 \pm 18.2	17.4 \pm 1.0	50.4 \pm 4.9
iris	15.4 \pm 12.8	19.0 \pm 8.2	74.2 \pm 6.9
led24	36.9 \pm 9.4	76.8 \pm 4.9	54.2 \pm 1.3
lymphography	39.7 \pm 15.3	28.6 \pm 2.8	66.8 \pm 2.3
monks-2	926.2 \pm 515.1	59.3 \pm 8.3	27.6 \pm 2.2
mplex-6	0.8 \pm 0.8	11.5 \pm 1.8	37.6 \pm 2.2
promoter	26.9 \pm 6.2	7.8 \pm 0.4	64.8 \pm 1.6
soybean	1684.0 \pm 30.5	96.9 \pm 2.5	71.5 \pm 0.8
switzerland	182.8 \pm 16.9	72.2 \pm 4.5	85.9 \pm 4.1
tictactoe	5792.5 \pm 236.0	241.6 \pm 14.0	29.6 \pm 1.1
va	351.2 \pm 28.8	123.0 \pm 7.0	70.5 \pm 1.9
votes	22.3 \pm 1.3	14.3 \pm 1.4	46.3 \pm 3.4
waveform	346.6 \pm 105.3	43.6 \pm 4.9	79.5 \pm 1.9
wine	14.2 \pm 4.6	16.5 \pm 3.6	81.3 \pm 4.0
zoo	3.1 \pm 0.7	19.0 \pm 1.2	75.8 \pm 3.0

CPU and memory costs are indicated in Table 4. Computationally, the CLEF algorithm is more costly than C4.5. Memory costs are not large. The table presents the memory requirements of the resulting classifier in terms of the total number of features present in the machine. CLEF typically constructs a small set of features, each of which consists of a simple bit pattern and a single weight. In order to measure the degree of overlap of the features that form a classifier, the average percentage of features matching an instance was evaluated. The "match train" column shows this measure for the instances in the training set. The values show that there is a high degree of overlap in the features that are constructed.

7 Related work

A variety of constructive methods have been devised for classification problems. A large class of algorithms construct networks of thresholded logic units, by adding boundaries that correct for misclassified examples (Parekh et. al, 1997). These algorithms also separate consistently labelled examples. The experimental results that have been published regarding these algorithms are limited, so they do not provide a good basis for comparison with CLEF.

Several algorithms that automatically construct a neural network configuration have also been used in classification tasks. Fahlman and Lebiere's (1990) cascade correlation method constructs a new hidden unit (feature) in order to minimize the residual error and freezes its defining

weights. The original input variables and the newly constructed unit become the input variables for the next layer. The algorithm has produced good results when applied to classification tasks. Wynne-Jones (1992) presents an approach called *node splitting* that detects when the hyperplane of a hidden unit is oscillating, indicating that the unit is being pushed in conflicting directions in feature space. Such a unit is split into two units, and the weights are set so that the units are moved apart from each other along an advantageous axis. A meiosis network (Hanson, 1990) is a feed-forward network in which the variance of each weight is maintained. For a hidden unit (feature) that has one or more weights of high variance, the unit is split into two. The input weights that define the feature, and the output weight for the linear combination are altered so that the two units are moved away from their means in opposite directions.

Support Vector Machines (Vapnik, 1995) can also be viewed as constructing features automatically, but the form of the features that are constructed needs to be defined a priori. More work would be needed to explore the relationship between CLEF and support vector machines.

8 Summary

CLEF is a classification algorithm that constructs a Φ -machine to fit the multiclass data. By using the ELF function approximator, non-linear features are constructed as needed. The sequence of feature sets produced by ELF has the effect that CLEF produces a sequence of Φ -machine classifiers. This sequence will ultimately produce a Φ -machine that separates the instances, whether or not they are linearly separable in the input variables. By contrast to decision trees, which recursively partition the training instances, CLEF constructs a classifier using the whole training set. This approach provides an advantage in terms of the accuracy of the resulting classifiers.

Acknowledgements

The authors thank Margie Connell and three anonymous reviewers for helpful comments on the previous drafts of this paper. This work was supported by NSF grant IRI-9711239. Doina Precup also acknowledges the support of the Fulbright foundation.

References

- Duda, R. O. & Hart, P. E. (1973). *Pattern classification and scene analysis*. Wiley & Sons, New York.
- Elomaa, T. & Rousu, J. (1996). Finding optimal multi-splits for numerical attributes in decision tree learning. Technical Report NC-TR-96-041, NeuroCOLT.
- Fahlman, S. E. & Lebiere, C. (1990). The cascade correlation architecture. In *Advances in Neural Information Processing Systems*, Volume 2 (pp. 524–532).
- Fayyad, U. & Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1022–1027). Morgan Kaufman, San Mateo, CA.
- Frean, M. (1990). *Small nets and short paths: Optimising neural computation*. PhD thesis, Center for Cognitive Science, University of Edinburgh.
- Fulton, T., Kasif, S. & Salzberg, S. (1995). Efficient algorithms for finding multi-way splits for decision trees. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 244–251). Morgan Kaufman.
- Hanson, S. J. (1990). Meiosis networks. In *Advances in Neural Information Processing Systems*, Volume 2 (pp. 533–541).
- Murphy, P. M. & Aha, D. W. (1994). *UCI repository of machine learning databases*. University of California, Irvine, CA: Department of Information and Computer Science.
- Nilsson, N. J. (1965). *Learning machines*. McGraw-Hill, New York.
- Parekh, R., Yang, J. & Honavar, V. (1997). Constructive neural network learning algorithms for multi-category real-valued pattern classification. Technical report, Iowa State University, Computer Science Department.
- Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufman, San Mateo, CA.
- Utgoff, P. E. & Precup, D. (1998). Constructive function approximation. In H. Motoda & H. Liu (Eds.), *Feature extraction, construction, and selection: A data-mining perspective*. Kluwer.
- Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer Verlag, New York.
- Wynne-Jones, M. (1992). Node splitting: A constructive algorithm for feed-forward neural networks. In *Advances in Neural Information Processing Systems* (pp. 1072–1079).

The Case Against Accuracy Estimation for Comparing Induction Algorithms

Foster Provost
Bell Atlantic Science and Tech
400 Westchester Avenue
White Plains, NY 10604
foster@basit.com

Tom Fawcett
Bell Atlantic Science and Tech
400 Westchester Avenue
White Plains, NY 10604
fawcett@basit.com

Ron Kohavi
Silicon Graphics Inc. M/S 8U-876
2011 N. Shoreline Blvd.
Mountain View, CA 94043
ronnyk@sgi.com

Abstract

We analyze critically the use of classification accuracy to compare classifiers on natural data sets, providing a thorough investigation using ROC analysis, standard machine learning algorithms, and standard benchmark data sets. The results raise serious concerns about the use of accuracy for comparing classifiers and draw into question the conclusions that can be drawn from such studies. In the course of the presentation, we describe and demonstrate what we believe to be the proper use of ROC analysis for comparative studies in machine learning research. We argue that this methodology is preferable both for making practical choices and for drawing scientific conclusions.

analysis is the proper methodology to provide such verification. We then provide a thorough analysis of classifier performance using standard machine learning algorithms and standard benchmark data sets. The results raise serious concerns about the use of accuracy, both for practical comparisons and for drawing scientific conclusions, even when predictive performance is the only concern.

The contribution of this paper is two-fold. We analyze critically a common assumption of machine learning research, provide insights into its applicability, and discuss the implications. In the process, we describe what we believe to be a superior methodology for the evaluation of induction algorithms on natural data sets. Although ROC analysis certainly is not new, for machine learning research it should be applied in a principled manner geared to the specific conclusions machine learning researchers would like to draw. We hope that this work makes significant progress toward that goal.

1 INTRODUCTION

Substantial research has been devoted to the development and analysis of algorithms for building classifiers, and a necessary part of this research involves comparing induction algorithms. A common methodology for such evaluations is to perform statistical comparisons of the accuracies of learned classifiers on suites of benchmark data sets. Our purpose is not to question the statistical tests (Dietterich, 1998; Salzberg, 1997), but to question the use of accuracy estimation itself. We believe that since this is one of the primary scientific methodologies of our field, it is important that we (as a scientific community) cast a critical eye upon it.

The two most reasonable justifications for comparing accuracies on natural data sets require empirical verification. We argue that a particular form of ROC

2 JUSTIFYING ACCURACY COMPARISONS

We consider induction problems for which the intent in applying machine learning algorithms is to build from the existing data a model (a *classifier*) that will be used to classify previously unseen examples. We limit ourselves to predictive performance—which is clearly the intent of most accuracy-based machine learning studies—and do not consider issues such as comprehensibility and computational performance.

We assume that the true distribution of examples to which the classifier will be applied is not known in advance. To make an informed choice, performance must be estimated using the data available. The different methodologies for arriving at these estimations have been described elsewhere (Kohavi, 1995;

Dietterich, 1998). By far, the most commonly used performance metric is classification accuracy.

Why should we care about comparisons of accuracies on benchmark data sets? Theoretically, over the universe of induction algorithms no algorithm will be superior on all possible induction problems (Wolpert, 1994; Schaffer, 1994). The tacit reason for comparing classifiers on natural data sets is that these data sets represent problems that systems might face in the real world, and that superior performance on these benchmarks may translate to superior performance on other real-world tasks. To this end, the field has amassed an admirable collection of data sets from a wide variety of classifier applications (Merz and Murphy, 1998). Countless research results have been published based on comparisons of classifier accuracy over these benchmark data sets. We argue that comparing accuracies on our benchmark data sets says little, if anything, about classifier performance on real-world tasks.

Accuracy maximization is not an appropriate goal for many of the real-world tasks from which our natural data sets were taken. Classification accuracy assumes equal misclassification costs (for false positive and false negative errors). This assumption is problematic, because for most real-world problems one type of classification error is much more expensive than another. This fact is well documented, primarily in other fields (statistics, medical diagnosis, pattern recognition and decision theory). As an example, consider machine learning for fraud detection, where the cost of missing a case of fraud is quite different from the cost of a false alarm (Fawcett and Provost, 1997).

Accuracy maximization also assumes that the class distribution (class priors) is known for the target environment. Unfortunately, for our benchmark data sets, we often do not know whether the existing distribution is the natural distribution, or whether it has been stratified. The iris data set has exactly 50 instances of each class. The splice junction data set (DNA) has 50% donor sites, 25% acceptor sites and 25% non-boundary sites, even though the natural class distribution is very skewed: no more than 6% of DNA actually codes for human genes (Saitta and Neri, 1998). Without knowledge of the target class distribution we cannot even claim that we are indeed maximizing accuracy for the problem from which the data set was drawn.

If accuracy maximization is not appropriate, why would we use accuracy estimates to compare induction algorithms on these data sets? Here are what we

believe to be the two best candidate justifications.

1. The *classifier* with the highest accuracy may very well be the classifier that minimizes cost, particularly when the classifier's tradeoff between true positive predictions and false positives can be tuned. Consider a learned model that produces probability estimates; these can be combined with prior probabilities and cost estimates for decision-analytic classifications. If the model has high classification accuracy because it produces very good probability estimates, it will also have low cost for any target scenario.
2. The *induction algorithm* that produces the highest accuracy classifiers may also produce minimum-cost classifiers by training it differently. For example, Breiman et al. (1984) suggest that altering the class distribution will be effective for building cost-sensitive decision trees (see also other work on cost-sensitive classification (Turney, 1996)).

To criticize the practice of comparing machine learning algorithms based on accuracy, it is not sufficient merely to point out that accuracy is not the metric by which real-world performance will be measured. Instead, it is necessary to analyze whether these candidate justifications are well founded.

3 ARE THESE JUSTIFICATIONS REASONABLE?

We first discuss a commonly cited special case of the second justification, arguing that it makes too many untenable assumptions. We then present the results of an empirical study that leads us to conclude that these justifications are questionable at best.

3.1 CAN WE DEFINE AWAY THE PROBLEM?

In principle, for a two-class problem one can repropportion ("stratify") the classes based on the target costs and class distribution. Once this has been done, maximizing accuracy on the transformed data corresponds to minimizing costs on the target data (Breiman et al., 1984). Unfortunately, this strategy is impracticable for conducting empirical research based on our benchmark data sets. First, the transformation is valid only for two-class problems. Whether it can be approximated effectively for multiclass problems is an open question.

Second, we do not know appropriate costs for these data sets and, as noted by many applied researchers (Bradley, 1997; Catlett, 1995; Provost and Fawcett, 1997), assigning these costs precisely is virtually impossible. Third, as described above, generally we do not know whether the class distribution in a natural data set is the "true" target class distribution.

Because of these uncertainties we cannot claim to be able to transform these cost-minimization problems into accuracy-maximization problems. Moreover, in many cases specifying target conditions is not just virtually impossible, it is actually impossible. Often in real-world domains there are no "true" target costs and class distribution. These change from time to time, place to place, and situation to situation (Fawcett and Provost, 1997).

Therefore the ability to transform cost minimization into accuracy maximization does not, by itself, justify limiting our comparisons to classification accuracy on the given class distribution. However, it may be that comparisons based on classification accuracy are useful because they are indicative of a broader notion of "better" performance.

3.2 ROC ANALYSIS AND DOMINATING MODELS

We now investigate whether an algorithm that generates high-accuracy classifiers is generally better because it also produces low-cost classifiers for the target cost scenario. Without target cost and class distribution information, in order to conclude that the classifier with higher accuracy is the better classifier, one must show that it performs better for any reasonable assumptions. We limit our investigation to two-class problems because the analysis is straightforward.

The evaluation framework we choose is Receiver Operating Characteristic (ROC) analysis (Egan, 1975; Swets and Pickett, 1982; Swets, 1988), a classic methodology from signal detection theory that is now common in medical diagnosis (Beck and Schultz, 1986) and has recently begun to be used more generally in AI (Bradley, 1997; Provost and Fawcett, 1997).

We briefly review some of the basics of ROC analysis. *ROC space* denotes the coordinate system used for visualizing classifier performance. In ROC space, typically the true positive rate, TP , is plotted on the Y axis and the false positive rate, FP , is plotted on the X axis. Each classifier is represented by the point in ROC space corresponding to its (FP, TP) pair. For models that produce a continuous output (*e.g.*, an estimate of

the posterior probability of an instance's class membership), these statistics vary together as a threshold on the output is varied between its extremes, with each threshold value defining a classifier. The resulting curve, called the *ROC curve*, illustrates the error tradeoffs available with a given model. ROC curves describe the predictive behavior of a classifier *independent of class distributions or error costs*, so they decouple classification performance from these factors.

For our purposes, a crucial notion is whether one model *dominates* in ROC space, meaning that all other ROC curves are beneath it or equal to it. A dominating model (*e.g.*, model NB in Figure 1a) is at least as good as all other models for all possible cost and class distributions. Therefore, if a dominating model exists, it can be considered to be the "best" model in terms of predictive performance. If a dominating model does not exist (as in Figure 1b), then none of the models represented is best under all target scenarios; in such cases, there exist scenarios for which the model that maximizes accuracy (or any other single-number metric) does not have minimum cost.

Figure 1 shows test-set ROC curves on two of the UCI domains from the study described below. Note the "bumpiness" of the ROC curves in Figure 1b (these were two of the largest domains with the least bumpy ROC curves). This bumpiness is typical of induction studies using ROC curves generated from a hold-out test set. As with accuracy estimates based on a single hold-out set, these ROC curves may be misleading because we cannot tell how much of the observed variation is due to the particular training/test partition. Thus it is difficult to draw strong conclusions about the expected behavior of the learned models. We would like to conduct ROC analysis using cross-validation.

Bradley (1997) produced ROC curves from 10-fold cross validation, but they are similarly bumpy. Bradley generated the curves using a technique known as *pooling*. In pooling, the i th points making up each raw ROC curve are averaged. Unfortunately, as discussed by Swets and Pickett (1982), pooling assumes that the i th points from all the curves are actually estimating the same point in ROC space, which is doubtful given Bradley's method of generating curves.¹ For our study it is important to have a good approximation of the expected ROC curve.

We generate results from 10-fold cross-validation using a different methodology, called *averaging*. Rather than

¹Bradley acknowledges this fact, and it is not germane to his study. However, it is problematic for us.

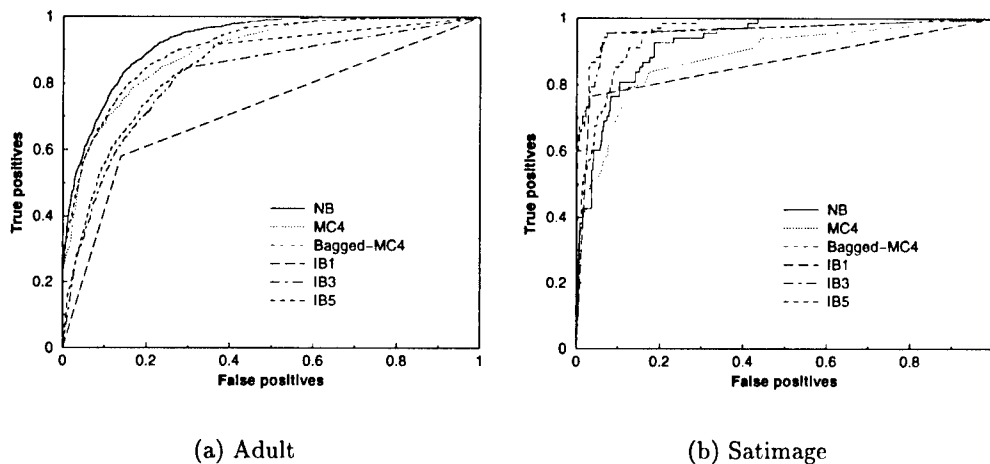


Figure 1: Raw (un-averaged) ROC curves from two UCI database domains

using the averaging procedure recommended by Swets and Pickett, which assumes normal-fitted ROC curves in a binormal ROC space, we average the ROC curves in the following manner. For k -fold cross-validation, the ROC curve from each of the k folds is treated as a function, R_i , such that $TP = R_i(FP)$. This is done with linear interpolations between points in ROC space² (if there are multiple points with the same FP , the one with the maximum TP is chosen). The averaged ROC curve is the function $\hat{R}(FP) = \text{mean}(R_i(FP))$. To plot averaged ROC curves we sample from \hat{R} at 100 points regularly spaced along the FP -axis. We compute confidence intervals of the mean of TP using the common assumption of a binomial distribution.

3.3 DO STANDARD METHODS PRODUCE DOMINATING MODELS?

We can now state precisely a basic hypothesis to be investigated: *Our standard learning algorithms produce dominating models for our standard benchmark data sets.* If this hypothesis is true (generally), we might conclude that the algorithm with higher accuracy is generally better, regardless of target costs or priors.³

²Note that classification performance anywhere along a line segment connecting two ROC points can be achieved by randomly selecting classifications (weighted by the interpolation proportion) from the classifiers defining the endpoints.

³However, even this conclusion has problems. Accuracy comparisons may select a non-dominating classifier because it is indistinguishable at the point of comparison—yet it

If the hypothesis is not true, then such a conclusion will have to rely on a different justification. We now provide an experimental study of this hypothesis, designed as follows.

From the UCI repository we chose ten datasets that contained at least 250 instances, but for which the accuracy for decision trees was less than 95% (because the ROC curves are difficult to read at very high accuracies). For each domain, we induced classifiers for the minority class (for Road we chose the class Grass). We selected several inducers from *MCC++* (Kohavi *et al.*, 1997): a decision tree learner (MC4), Naive Bayes with discretization (NB), k -nearest neighbor for several k values (IB k), and Bagged-MC4 (Breiman, 1996). MC4 is similar to C4.5 (Quinlan, 1993); probabilistic predictions are made by using a Laplace correction at the leaves. NB discretizes the data based on entropy minimization (Dougherty *et al.*, 1995) and then builds the Naive-Bayes model (Domingos and Pazzani, 1997). IB k votes the closest k neighbors; each neighbor votes with a weight equal to one over its distance from the test instance.

The averaged ROC curves are shown in Figures 2 and 3. For *only one* (Vehicle) of these ten domains was there an absolute dominator. In general, very few of the 100 runs we performed (10 data sets, 10 cross-validation folds each) had dominating classifiers. Some cases are very close, for example Adult and Waveform-21. In other cases a curve that dominates in one area of ROC space is dominated in another. Therefore, we may be much worse elsewhere.

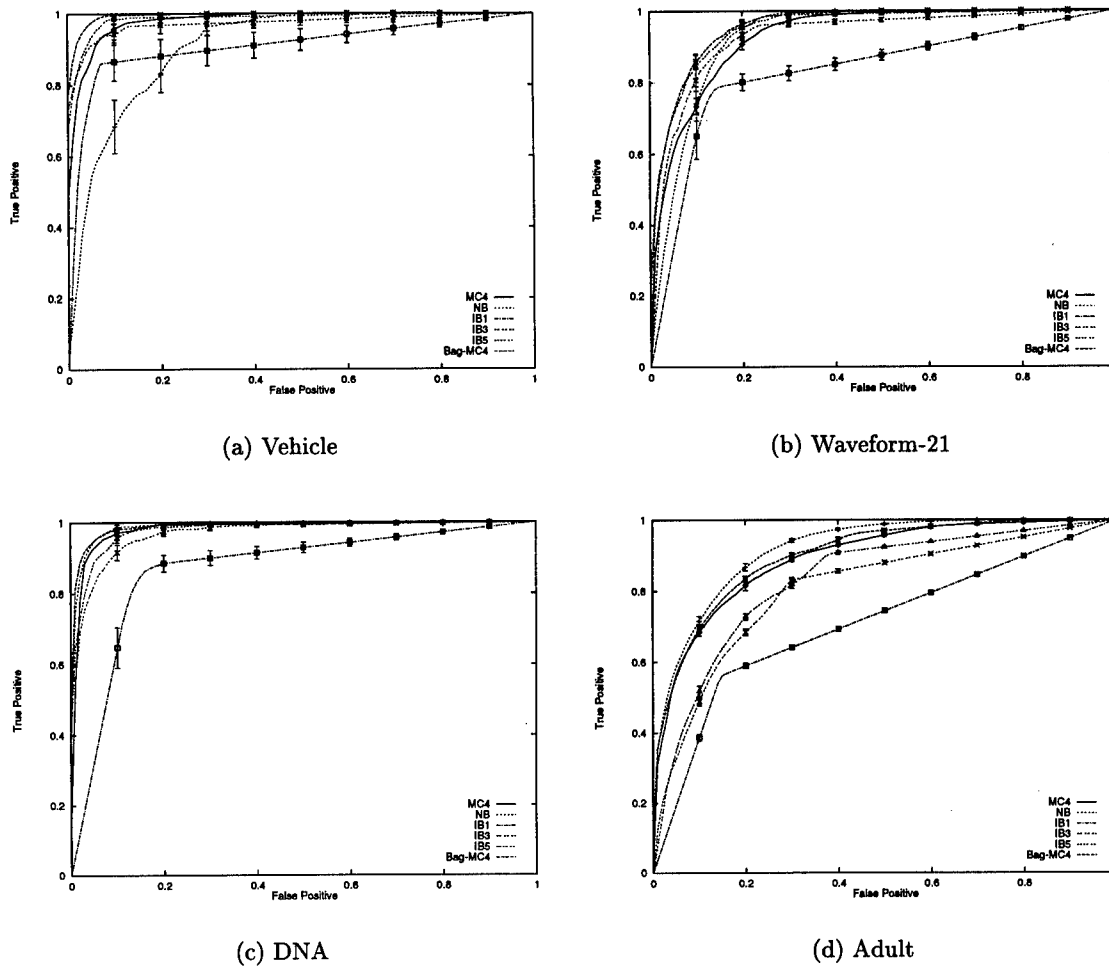


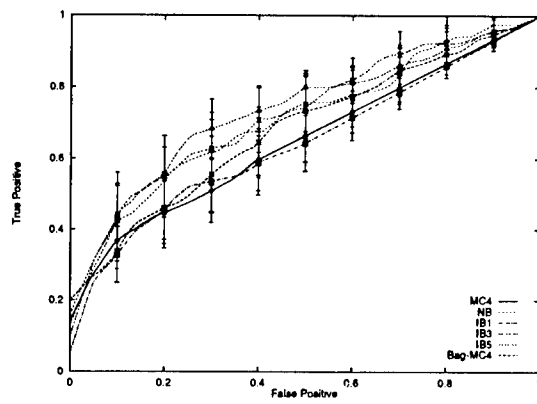
Figure 2: Smoothed ROC curves from UCI database domains

can refute the hypothesis that our algorithms produce (statistically significantly) dominating classifiers.

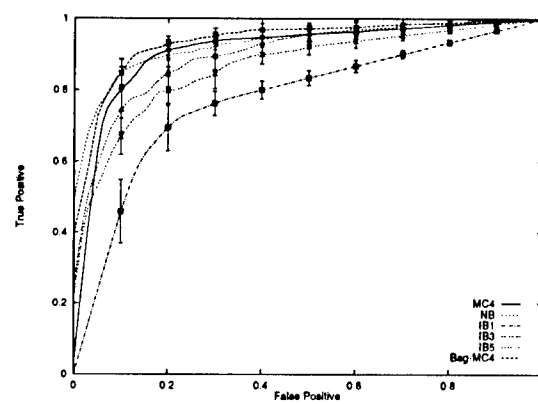
This draws into question claims of “algorithm A is better than algorithm B” based on accuracy comparison. In order to draw such a conclusion in the absence of target costs and class distributions, the ROC curve for algorithm A would have to be a significant dominator of algorithm B. This has obvious implications for machine learning research.

In practical situations, often a weaker claim is sufficient: Algorithm A is a good choice because it is at least as good as Algorithm B (*i.e.*, their accuracies are not significantly different). It is clear that this type of conclusion also is not justified. In many domains, curves that are statistically indistinguishable

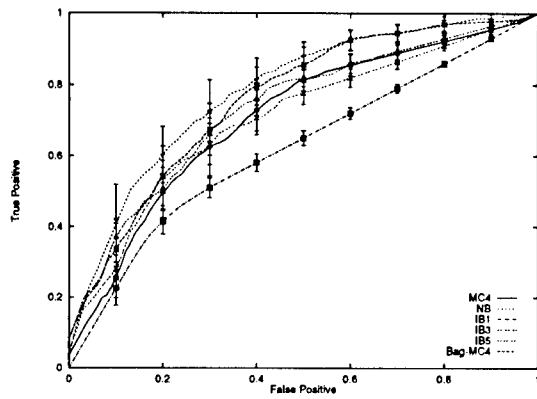
from dominators in one area of the space are significantly dominated in another. Moreover, in practical situations typically comparisons are not made with the wealth of classifiers we are considering. More often only a few classifiers are compared. Considering general pairwise comparisons of algorithms, there are many cases where each model in a pair is clearly much better than the other in different regions of ROC space. This clearly draws into question the use of single number metrics for practical algorithm comparison, unless these metrics are based on precise target cost and class distribution information.



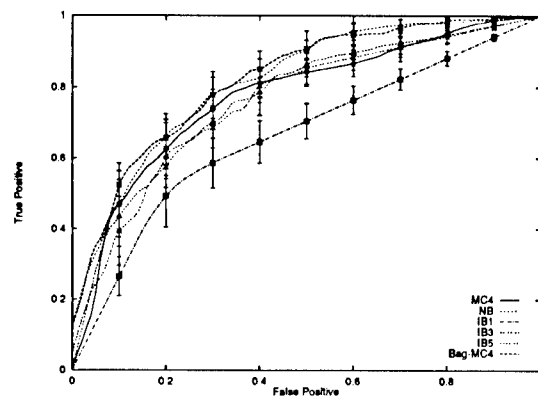
(a) Breast cancer



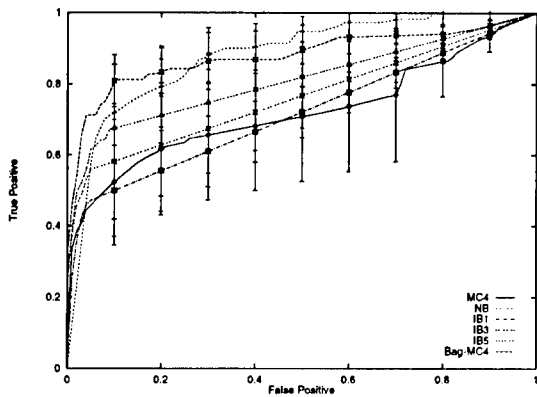
(b) CRX



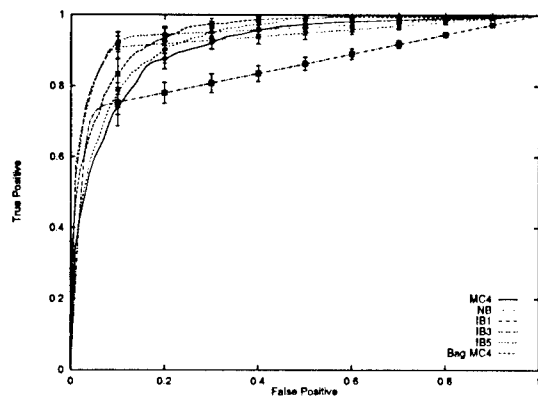
(c) German



(d) Pima



(e) RoadGrass



(f) Satimage

Figure 3: Smoothed ROC curves from UCI database domains, cont'd

3.4 CAN STANDARD METHODS BE COERCED TO YIELD DOMINATING ROC CURVES?

The second justification for using accuracy to compare algorithms is subtly different from the first. Specifically, it allows for the possibility of coercing algorithms to produce different behaviors under different scenarios (such as in cost-sensitive learning). If this can be done well, accuracy comparisons are justified by arguing that for a given domain, the algorithm with higher accuracy will also be the algorithm with lower cost for all reasonable costs and class distributions.

Confirming or refuting this justification completely is beyond the scope of this paper, because how best to coerce algorithms for different environmental conditions is an open question. Even the straightforward method of stratifying samples has not been evaluated satisfactorily. We propose that the ROC framework outlined so far, with a minor modification, can be used to evaluate this question as well.

For algorithms that may produce different models under different cost and class distributions, the ROC methodology as stated above is not quite adequate. We must be able to evaluate the performance of the *algorithm*, not an individual model. However, one can characterize an algorithm's performance for ROC analysis by producing a composite curve for a set of generated models. This can be done using pooling, or by using the convex hull of the ROC curves produced by the set of models, as described in detail by Provost and Fawcett (1997; 1998).

We can now form a hypothesis for our second potential justification: *Our standard learning algorithms produce dominating ROC curves for our standard benchmark data sets.* Confirming this hypothesis would be an important step in justifying the common practice of ignoring target costs and class distributions in classifier comparisons on natural data. Unfortunately, we know of no confirming evidence.

On the other hand, there is disconfirming evidence. First, consider the results presented above. Naive Bayes is robust with respect to changes in costs—it will produce the same ROC curve regardless of the target costs and class distribution. Furthermore, it has been shown that decision trees are surprisingly robust if the probability estimates are generated with the Laplace estimate (Bradford *et al.*, 1998). If this result holds generally, the results in the previous section would disconfirm the present hypothesis as well.

Second, Bradley's (1997) results provide disconfirming evidence. Specifically, he studied six real-world medical data sets (four from the UCI repository and two from other sources). Bradley plotted the ROC curves of six classifier learning algorithms, consisting of two neural nets, two decision trees and two statistical techniques. Bradley uses composite ROC curves formed by training models differently for different cost distributions. We have previously criticized the design of his study for the purpose of answering our question. However, if the results can be replicated under the current methodology, they would make a strong statement. *Not one* of the six data sets had a dominating classifier. This implies that for each domain there exist disjoint sets of conditions for which different induction algorithms are preferable.

4 RECOMMENDATIONS AND LIMITATIONS

When designing comparative studies, researchers should be clear about the conclusions they want to be able to draw from the results. We have argued that comparisons of algorithms based on accuracy are unsatisfactory when there is no dominating classifier. However, presenting the case against the use of accuracy is only one of our goals. We also want to show how precise comparisons still can be made, even when the target cost and class distributions are not known.

If there is no dominator, conclusions must be qualified. No single number metric can be used to make very strong conclusions without domain-specific information. However, it is possible to look at ranges of costs and class distributions for which each classifier dominates. The problems of cost-sensitive classification and learning with skewed class distributions can be analyzed precisely.

Even without knowledge of target conditions, a precise, concise, robust specification of classifier performance can be made. As described in detail by Provost and Fawcett (1997), the slopes of the lines tangent to the ROC convex hull determine the ranges of costs and class distributions for which particular classifiers minimize cost. For specific target conditions, the corresponding slope is the cost ratio times the reciprocal of the class ratio. For our ten domains, the optimal classifiers for different target conditions are given in Table 1. For example, in the Road domain (see Figure 3 and Table 1), Naive Bayes is the best classifier for any target conditions corresponding to a slope less than 0.38, and Bagged-MC4 is best for slopes greater

Table 1: Locally dominating classifiers for ten UCI domains

Domain	Slope range	Dominator	Domain	Slope range	Dominator
Adult	[0, 7.72]	NB	Pima	[0, 0.06]	NB
	[7.72, 21.6]	Bagged-MC4		[0.06, 0.11]	Bagged-MC4
	[21.6, ∞)	NB		[0.11, 0.30]	NB
Breast cancer	[0, 0.37]	NB		[0.30, 0.82]	Bagged-MC4
	[0.37, 0.5]	IB3		[0.82, 1.13]	NB
	[0.5, 1.34]	IB5		[1.13, 4.79]	Bagged-MC4
	[1.34, 2.38]	IB3		[4.79, ∞)	NB
	[2.38, ∞)	Bagged-MC4	Satimage	[0, 0.05]	NB
CRX	[0, 0.03]	Bagged-MC4		[0.05, 0.22]	Bagged-MC4
	[0.03, 0.06]	NB		[0.22, 2.60]	IB5
	[0.06, 2.06]	Bagged-MC4		[2.60, 3.11]	IB3
	[2.06, ∞)	NB		[3.11, 7.54]	IB5
German	[0, 0.21]	NB		[7.54, 31.14]	IB3
	[0.21, 0.47]	Bagged-MC4		[31.14, ∞)	Bagged-MC4
	[0.47, 3.08]	NB	Waveform 21	[0, 0.25]	NB
	[3.08, ∞)	IB5		[0.25, 4.51]	Bagged-MC4
Road (Grass)	[0, 0.38]	NB		[4.51, 6.12]	IB5
	[0.38, ∞)	Bagged-MC4		[6.12, ∞)	Bagged-MC4
DNA	[0, 1.06]	NB	Vehicle	[0, ∞)	Bagged-MC4
	[1.06, ∞)	Bagged-MC4			

than 0.38. They perform equally well at 0.38. We admit that this is not as elegant as a single-number comparison, but we believe it to be much more useful, both for research and in practice.

In summary, if a dominating classifier does not exist and cost and class distribution information is unavailable, no strong statement about classifier superiority can be made. However, one might be able to make precise statements of superiority for specific regions of ROC space. For example, if all you know is that few false positive errors can be tolerated, you may be able to find a particular algorithm that is superior at the "far left" edge of ROC space.

We limited our investigation to two classes. This does not affect our conclusions since our results are negative. However, since we are also recommending an analytical framework, we note that extending our work to multiple dimensions is an interesting open problem.

Finally, we are not completely satisfied with our method of generating confidence intervals. The present intervals are appropriate for the Neyman-Pearson observer (Egan, 1975), which wants to maximize TP for a given FP. However, their appropriateness is questionable for evaluating minimum expected cost, for which a given set of costs contours ROC space with lines of a particular slope. Although this is an area of future work, it is not a fundamental drawback to the methodology.

5 CONCLUSIONS

We have offered for debate the justification for the use of accuracy estimation as the primary metric for comparing algorithms on our benchmark data sets. We have elucidated what we believe to be the top candidates for such a justification, and have shown that either they are not realistic because we cannot specify cost and class distributions precisely, or they are not supported by experimental evidence.

We draw two conclusions from this work. First, the justifications for using accuracy to compare classifiers are questionable at best. Second, we have described what we believe to be the proper use of ROC analysis as applied to comparative studies in machine learning research. ROC analysis is not as simple as comparing with a single-number metric. However, we believe that the additional power it delivers is well worth the effort. In certain situations, ROC analysis allows very strong, general conclusions to be made—both positive and negative. In situations where strong, general conclusions cannot be made, ROC analysis allows very precise analysis to be conducted.

Although ROC analysis is not new, in machine learning research it has not been applied in a principled manner, geared to the specific conclusions machine learning researchers would like to draw. We hope that this work makes significant progress toward that goal.

Acknowledgements

We thank the many with whom we have discussed the justifications for accuracy-based comparisons and ROC analysis as applied to classifier learning. Rob Holte provided very helpful comments on a draft of this paper.

References

- J. R. Beck and E. K. Schultz. (1986) The use of ROC curves in test performance evaluation. *Arch Pathol Lab Med*, 110:13–20.
- J. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. Brodley. (1998) Pruning decision trees with misclassification costs. In *Proceedings of ECML-98*, pages 131–136.
- A. P. Bradley. (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159.
- L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. (1984) *Classification and Regression Trees*. Wadsworth International Group.
- L. Breiman. (1996) Bagging predictors. *Machine Learning*, 24:123–140.
- J. Catlett. (1995) Tailoring rulesets to misclassification costs. In *Proceedings of the 1995 Conference on AI and Statistics*, pages 88–94.
- T. G. Dietterich. (1998) Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*. To appear.
- P. Domingos and M. Pazzani. (1997) Beyond independence: Conditions for the optimality of the simple Bayesian classifier. *Machine Learning*, 29:103–130.
- J. Dougherty, R. Kohavi, and M. Sahami. (1995) Supervised and unsupervised discretization of continuous features. In A. Priditis and S. Russell, (eds.), *Proceedings of ICML-95*, pages 194–202. Morgan Kaufmann.
- J. P. Egan. (1975) *Signal Detection Theory and ROC Analysis*. Series in Cognition and Perception. Academic Press, New York.
- T. Fawcett and F. Provost. (1997) Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3). Available: <http://www.croftj.net/~fawcett/DMKD-97.ps.gz>.
- R. Kohavi, D. Sommerfield, and J. Dougherty. (1997) Data mining using *MLC++*: A machine learning library in C++. *International Journal on Artificial Intelligence Tools*, 6(4):537–566. Available: <http://www.sgi.com/Technology/mlc>.
- R. Kohavi. (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In C. S. Mellish, (ed.), *Proceedings of IJCAI-95*, pages 1137–1143. Morgan Kaufmann. Available: <http://robotics.stanford.edu/~ronnyk>.
- C. Merz and P. Murphy. (1998) UCI repository of machine learning databases. Available: <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- F. Provost and T. Fawcett. (1997) Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proceedings of KDD-97*, pages 43–48. AAAI Press.
- F. Provost and T. Fawcett. (1998) Robust classification systems for imprecise environments. In *Proceedings of AAAI-98*. AAAI Press. To appear. Available: <http://www.croftj.net/~fawcett/papers/aaai98-dist.ps.gz>.
- J. R. Quinlan. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California.
- L. Saitta and F. Neri. (1998) Learning in the “Real World”. *Machine Learning*, 30:133–163.
- S. L. Salzberg. (1997) On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:317–328.
- C. Schaffer. (1994) A conservation law for generalization performance. In *ICML-94*, pages 259–265. Morgan Kaufmann.
- J. A. Swets and R. M. Pickett. (1982) *Evaluation of Diagnostic Systems: Methods from Signal Detection Theory*. New York: Academic Press.
- J. Swets. (1988) Measuring the accuracy of diagnostic systems. *Science*, 240:1285–1293.
- P. Turney. (1996) Cost sensitive learning bibliography. Available: <http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html>.
- D. H. Wolpert. (1994) The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework. In D. H. Wolpert, (ed.), *The Mathematics of Generalization*. Addison Wesley.

Theory Refinement for Bayesian Networks with Hidden Variables

Sowmya Ramachandran,
 Stottler Henke and Associates, Inc.,
 1660, So. Amphlett Blvd. Ste. 350,
 San Mateo, CA, 94402
 sowmya@shai.com

Raymond J. Mooney
 Department of Computer Sciences,
 University of Texas at Austin,
 Austin, TX, 78712
 mooney@cs.utexas.edu

Abstract

While there has been a growing interest in the problem of learning Bayesian networks from data, no technique exists for learning or revising Bayesian networks with hidden variables (i.e. variables not represented in the data), that has been shown to be efficient, effective, and scalable through evaluation on real data. The few techniques that exist for revising such networks perform a blind search through a large space of revisions, and are therefore computationally expensive. This paper presents BANNER, a technique for using data to revise a given Bayesian network with noisy-or and noisy-and nodes, to improve its classification accuracy. The initial network can be derived directly from a logical theory expressed as propositional rules. BANNER can revise networks with hidden variables, and add hidden variables when necessary. Unlike previous approaches, BANNER employs mechanisms similar to logical theory refinement techniques for using the data to focus the search for effective modifications. Experiments on real-world problems in the domain of molecular biology demonstrate that BANNER can effectively revise fairly large networks to significantly improve their accuracies.

1 Introduction

Bayesian networks have become the most popular approach to uncertain reasoning due to their precise probabilistic semantics as well their success in practical applications. In an attempt to automate their construction, induction of Bayes nets has become a topic of increasing interest. A number of learning methods have been developed for the case where all relevant variables are observable (Heckerman, 1995). Parameter learning methods for networks with *hidden variables* (variables not represented in the data) have also been developed (Russell, Binder, Koller, & Kanazawa, 1995; Thiesson, 1995). However, learning both the structure and the parameters of a Bayesian network with hidden variables remains a problem. Many of the existing methods can be adapted to discover hidden variables, but only by conducting extensive search

that is impractical for most problems. A recent development is MS-EM (Friedman, 1997), which learns the structure of a network with hidden variables; however, it requires specifying the number of hidden variables and has not been tested on real data.

As demonstrated by *theory refinement* research on rule-bases, using empirical data to revise an initial imperfect knowledge base can significantly improve performance over induction from scratch (Opitz & Shavlik, 1993; Ourston & Mooney, 1994; Towell & Shavlik, 1994; Mahoney & Mooney, 1994; Brunk & Paz-zani, 1995). A few techniques have been developed for revising Bayesian networks (Lam & Bacchus, 1994; Buntine, 1991); however, they do not handle hidden variables. Many existing Bayes-net induction methods could be adapted to revision, but only by examining all possible individual modifications. By contrast, rule-revision systems use classification errors on the training data to propose specific modifications rather than blindly examining all possible options. The result is an efficient, directed revision process.

We have developed a technique, BANNER, for refining Bayesian networks with hidden variables that, like rule-refinement algorithms, uses the data to focus the search for effective modifications. BANNER's goal is to improve the accuracy of an initial network for a specific inference task by modifying both its parameters and structure, including adding new hidden variables. Although Bayesian networks can simultaneously support many types of inference, training directly for the desired classification task results in better performance (Friedman & Goldszmidt, 1996). Since general Bayesian networks are impractical for many large problems because the number of parameters grows exponentially in the fan-in of a node, we focus on networks with *noisy-or* and *noisy-and* nodes, specialized models that require only a linear number of param-

eters (Pearl, 1988; Pradhan, Provan, Middleton, & Henrion, 1994). Since these models are close to logical functions, they also allow a rule-base to be used as an initial theory by mapping the rules to a network in the obvious way. Existing results show that the accuracy of rule bases can be dramatically improved by mapping them to a representation that provides numerical summing of evidence (Towell & Shavlik, 1994; Mahoney & Mooney, 1994). However, the neural networks or certainty-factor rules employed in these results do not provide an interpretable knowledge base with parameters that have a precise semantics. An important goal of theory refinement is to provide interpretable knowledge, and we believe Bayes nets are preferable in this regard.

Experimental evaluation of Bayes net learning has largely been conducted on artificial data and not adequately compared to other methods on real problems (exceptions include Provan and Singh (1994), Friedman and Goldszmidt (1996)), and we know of no Bayes-net results on revising real knowledge bases to fit actual data. We have evaluated BANNER on several realistic problems used to test other theory refinement systems, obtaining performance competitive with the current best results while maintaining the advantages of a Bayes-net representation. The remainder of the paper presents an overview of BANNER's learning algorithm and the promising results of this evaluation.

2 Refinement Algorithm

As in general in theory refinement, the goal is to minimally modify the initial theory to make it consistent with the available training data. Taking the standard approach, BANNER employs one procedure to revise the parameters of a network and another to revise the structure. First, the parameters are revised to improve classification accuracy. If the resulting network does not adequately fit the training data, the structure of the network is modified and the parameters are retrained. This process repeats until it is determined that additional training results in over-fitting.¹ In this paper, we focus on structure revision. Our current implementation includes two parameter revision algorithms, BANNER-PR (Ramachandran & Mooney, 1996) and C-APN (based on (Russell et al., 1995)), which use different forms of gradient descent. Ramachandran (1998) presents further details.

¹The parameter revision component uses 10-fold internal cross-validation on the training set to determine when to stop (Mitchell, 1997).

Structure revision exploits the idea that networks with noisy-or/and nodes are similar to logical theories and therefore techniques used to revise rule bases are useful. These methods attribute classification errors on particular examples to specific portions of the theory and directly construct revisions to handle the misclassified cases. Most logical refinement systems use abduction to diagnose faults (Mooney, 1997). Since Bayesian networks place no restrictions on the direction of inference, abduction can be performed using the standard inference algorithms. In addition, *leak nodes* (Pradhan et al., 1994) provide a way to model the incompleteness and incorrectness of a Bayesian network with noisy-or/and nodes. A leak node is a source in the graph added as an extra input to a node in order to represent a possible unknown cause. BANNER diagnoses faults in a network by temporarily instrumenting each node with leak nodes that indicate potential revision points. It then uses training data to select a small set of revision points and construct appropriate refinements.

2.1 Selecting Revision Points

The procedure for instrumenting a network with leak nodes is best illustrated with an example, such as that shown in Figure 1 (A-G are the original nodes). Each noisy-or/and has an added parent called a *node-leak* node. In order to avoid significantly altering the semantics of the net, the prior of the leak node and its link parameter are initially set very low. However, when the algorithm detects misclassifications, it re-estimates the prior probabilities by training a copy of the network augmented with leak node-leak nodes using the parameter revision module. All of the original noisy-or (noisy-and) nodes also have their parents routed through an *intervening* noisy-and (noisy-or) node. The intervening nodes themselves have attached leak nodes called *link-leak* nodes. To avoid altering the semantics, the weights on the links are set to simulate logical functions and the prior probability of the link-leak node is set to the weight on the original link. The leak nodes effectively represent possible faults in the theory, with node-leak nodes representing the need for new inputs to a node, and link-leak nodes representing the need for new intervening hidden variables between two nodes.

Once the network is properly instrumented, BANNER performs abduction on each misclassified example to generate a set of repairs that could correct the example. This involves instantiating both the evidence and the target variables in the augmented network to

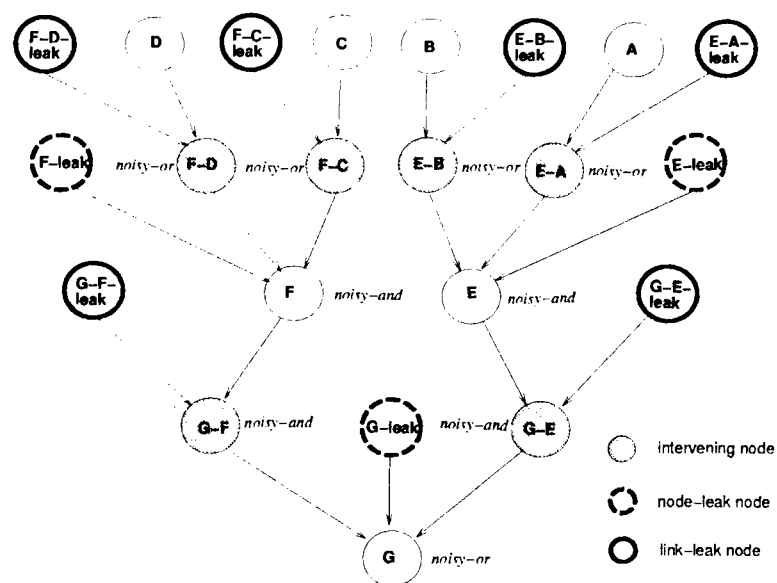


Figure 1: Augmenting a network with leak nodes

their observed values and inferring the beliefs associated with the leak nodes using standard Bayesian inference. For each misclassified example, it collects a set of leak nodes, whose beliefs deviate from their prior probability by more than 10%. Such leak nodes are said to *cover* the example, and indicate potential revision points in the theory. When the belief in the truth of a leak node decreases from its prior, it is called an *inhibitor* for that example; if it increases, it is called an *enabler*. Each leak node covering an example is associated with the degree to which its belief deviated from its prior, indicating the extent to which it is blamed for the misclassification. Once leak nodes are collected for all misclassified examples, BANNER uses a greedy set covering algorithm (where the contribution of each leak node is weighted by its degree) to generate a small set of leak nodes that cover all of the misclassified examples. While BANNER uses only misclassified examples to generate a set of revision points, it performs abduction on all the examples, generating leak nodes that are enablers or inhibitors for each example. This information is used during the generation of appropriate revisions.

2.2 Revision Operators

For each revision point in the covering set, BANNER implements one of the following modifications to help correct the misclassified examples covered by the corresponding leak node: 1) Add a new parent, 2) Add a new hidden node, 3) Delete a link. The first operator

is invoked when a revision point is a node-leak node, in which case it adds a new parent to the appropriate node in the original network. In the example, if *G-leak* is a selected revision point, then a new parent is added to *G*. The heuristic for selecting the new parent is discussed below.

If a revision point is a link-leak node, BANNER modifies the corresponding link. One option is to introduce a new hidden variable with an additional parent and the same type as the corresponding intervening node. In the example, if *E-A-leak* is the revision point, a new noisy-or node is added between *E* and *A* (see Figure 2). The rationale for such a revision is that the previous step of abduction with the augmented network indicated that such a structure would better explain the misclassified data.

However, in some cases, the problematic link is simply deleted. For example, if *E-A-leak* is an enabler for several examples but never an inhibitor, the link may be deleted to correct the misclassified examples without affecting other examples since the link is effectively an always-true input to a noisy-and which therefore has no effect. A dual argument can be made for noisy-or nodes. A link is also deleted if, when a hidden node is added, the chosen parent has the same effect as link deletion. For example, if the negation of *A* is chosen as the new parent of *E-A*, the link between *E* and *A* is deleted.

New parents are selected based on the examples for

Given: An initial network, and a set of training data. **Output:** A revised network.

1. Initialize the parameters of the network either randomly or based on some prior knowledge.
2. Repeat steps a-e until there is no improvement in training accuracy over a pre-specified number of consecutive cycles.
 - (a) set *train-net* = initial network.
 - (b) set *leak-net* = *train-net* augmented with *node-leak* nodes.
 - (c) Train network *train-net* to revise parameters.
 - (d) If the previous step indicates overfitting, or all examples are correctly classified, return *train-net*.
 - (e) else
 - i. Train network *leak-net* to estimate prior probabilities of the *node-leak* nodes.
 - ii. Set *augmented-net* = *train-net* augmented with *node-leak* and *link-leak* nodes.
 - iii. Copy priors of leak nodes from *leak-net* to *augmented-net*.
 - iv. For each example,
 - A. Instantiate input and target nodes of *augmented-net* with values from the example.
 - B. Infer beliefs of all the nodes in *augmented-net*.
 - C. Collect all enabled and inhibited *node-leak* and *link-leak* nodes.
 - v. Set *revision-points* = small set of *node-leak* and *link-leak* nodes that cover all the misclassified examples (computed using greedy set covering)
 - vi. For each revision point in *revision-points*, revise *train-net* at the revision point using one of the revision operators.

Figure 3: Outline of the Refinement Algorithm

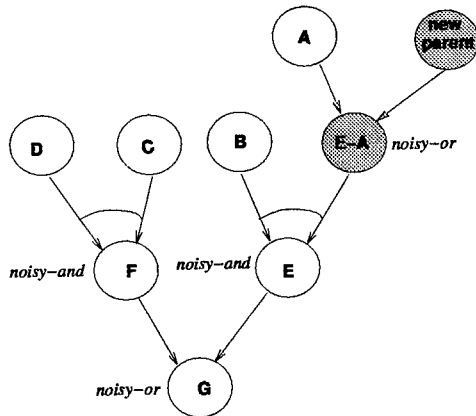


Figure 2: Revision operator: Adding a hidden node

which the chosen leak-node is an enabler or inhibitor. The new parent needs to be true for the examples it must enable and false for the ones it must inhibit. BANNER uses a standard *information gain* metric (Quinlan, 1990) to choose a parent that best discriminates between these two sets of examples. This metric, commonly used in inductive learning algorithms (Mahoney & Mooney, 1994; Quinlan, 1990, 1986), estimates the information gained about a target function value from knowing the value of an attribute. Two versions of this metric that are commonly used. The version used by Quinlan (1990) to learn propositional Horn-clause theories, is designed to pick a feature that best discriminates between sets of examples, with the additional constraint that the feature have

specific values (e.g. true or false) for each set of examples. This version is most appropriate for our theory refinement algorithm because we need to select a new parent that discriminates between the examples that need an enabling influence, and the examples that need an inhibitory influence, with the additional constraint that the new parent be true for the former set of examples and false for the latter set of examples.

Suppose that we are given a set of examples, S , of size N , of which N^+ are *positive* examples of a given class C , and N^- are *negative* examples of C . Also assume that all the features in the examples are boolean-valued. For any given feature F , let N_f be the number of examples for which F is true; of these let, N_f^+ be the number of examples which are positive examples of C , and N_f^- be the number of examples which are negative examples of C . Then, the reduction due to F in the total number of bits required to encode the positive members of C is given by

$$Gain(C, F) = N_f^+ * (I(S) - I(N_f)),$$

where $I(S) = -\log_2 \left(\frac{N^+}{N^+ + N^-} \right)$ is the number of bits required to encode a positive member of class C , and $I(N_f) = -\log_2 \left(\frac{N_f^+}{N_f^+ + N_f^-} \right)$ is the number of bits required to encode the positive members of class C , given that F is true. The higher the value of this function, the greater the correlation between the examples for which F is true and the positive examples of C . Note that this computation can be easily generalized to hidden variables and variables with missing values.

Information gain for such nodes can be obtained by weighting the frequency measures N_f^+ and N_f^- by the degree of belief associated with these nodes for each example.

So far, we have described this metric with a view to selecting an enabling parent. The same metric is used to select an inhibitory parent by defining N_f to be the number of examples for which F is false. Every other term in the computation of the metric is defined as before. In general, all nodes in the network and their negations are potential candidates; however, to avoid redundancy and the introduction of loops, the existing parents and descendents of the recipient of the new parent are excluded. Figure 3 shows a summary of the overall algorithm.

3 Experimental Evaluation

We conducted experiments on realistic problems and data to demonstrate that BANNER is effective at revising networks to improve their classification accuracy. We also compared its performance to naive Bayes which learns a simple Bayes net that includes all features and assumes conditional independence,² with KBANN (Towell & Shavlik, 1994) a neural-network refinement method, RAPTURE (Mahoney & Mooney, 1994) a certainty-factor refinement method, and with two standard inductive algorithms: C4.5 (Quinlan, 1993) for decision trees and BACKPROP (McClelland & Rumelhart, 1988) for neural networks. In order to study the contribution of BANNER's components, we also performed *ablation* studies, where we disabled parts of the algorithm and compared performance to the full system. BANNER-IND, is an inductive version which does not utilize an initial theory but starts with a default network with input and output variables but no links, and BANNER-PR (parameter revision), which uses an initial theory but does not perform structure revision. Finally, we specifically evaluated structure revision by attempting to fix an artificially corrupted initial theory.

We present results on two molecular biology problems employed in previous refinement experiments: recognizing promoters and splice-junctions in DNA strands (Towell & Shavlik, 1994). These problems include imperfect, expert-provided theories represented as propositional rules. These theories contain fan-ins of up to 17 inputs, which would require more than 130,000

parameters for general nodes, demonstrating the importance of using noisy-or/ands. Here we present the splice-junction results and results on a corrupted version of the promoter theory. BANNER also performs well on revising the original promoter theory, but since its structure is already adequate, this problem does not test structure revision. The system also performed well on revising a knowledge base on C++ programming to model students for an intelligent tutoring system (Baffes & Mooney, 1996). Ramachandran (1998) presents complete results.

In order to compare to previous results, we generated learning curves in which the data was randomly split into independent training and test sets, systems were trained on the training data, and then tested on classifying the test examples. Results were averaged over 20 random training/test splits. This was done for training sets with increasing number of examples. A two-tailed paired t-test is used to evaluate the statistical significance of differences in performance given a specific number of training examples.

3.1 DNA Splice-Junction

This problem addresses the task of detecting *splice-junctions*, the boundaries between the utilized and unutilized sequences in DNA. The data set consists of 3190 examples consisting of strings of 60 nucleotides with the values A, C, G, or T, and assigned to three different categories. The initial theory consists of 47 propositional rules.

Figure 4 shows the primary results and Figure 5 shows the ablation results. The experiment provides evidence that BANNER is successful at improving the accuracy of the initial theory significantly with just a small number of examples. The accuracy of the initial theory has risen from 55%, before revision, to 73.6% when trained on just 20 examples, and to about 91.2% when trained on 400 examples. The performance of the three refinement algorithms RAPTURE, BANNER, and KBANN are similar, although RAPTURE performs slightly better. The differences between RAPTURE and BANNER are small but statistically significant for all points on the learning curve at the 0.01 level. The inductive algorithms all perform significantly worse for smaller training sets, although NAIVE BAYES catches up with RAPTURE at 200 examples. The differences between the BANNER and NAIVE BAYES are significant at at least the 0.01 level for 20, 50, and 100 examples, where the former performs considerably better, at the 0.001 level for 400 examples where it performs slightly worse.

²Our version includes smoothing with Laplace estimates which significantly improves performance (Kohavi, Becker, & Sommerfield, 1997)

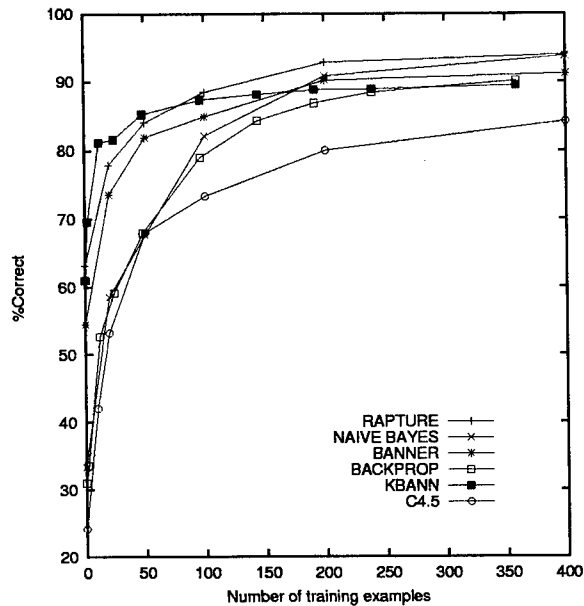


Figure 4: Splice-Junction: Performance of Various Systems

Figure 5 demonstrates that structure revision contributes significantly to BANNER's performance on smaller training sets. Structure revision has contributed to an improvement in accuracy of about 13% over BANNER-PR for 20 examples (significant at 0.001 level), and an improvement of about 2.8% for 50 examples (significant at the 0.05 level). The revisions that contributed the most to this improvement were deletions of the links between nodes *IE* and *PR*, and nodes *EI* and *P5G*. The differences between BANNER and BANNER-PR are not statistically significant at the rest of the points on the learning curve. As expected, starting out with an initial theory gives BANNER a significant edge over BANNER-IND. The difference in performance between these systems is statistically significant for all points on the learning curves, except at 100 example, at levels of at least 0.02.

3.2 Evaluation of Structure Revision on DNA Promoter

In order to more directly study structure revision, an existing theory with adequate structure was corrupted and BANNER's ability to recover the lost structure was examined. The DNA promoter recognition problem involves identifying DNA sequences that indicate the start of a new gene. Figure 6 shows a portion of the Bayesian network derived from the initial theory for

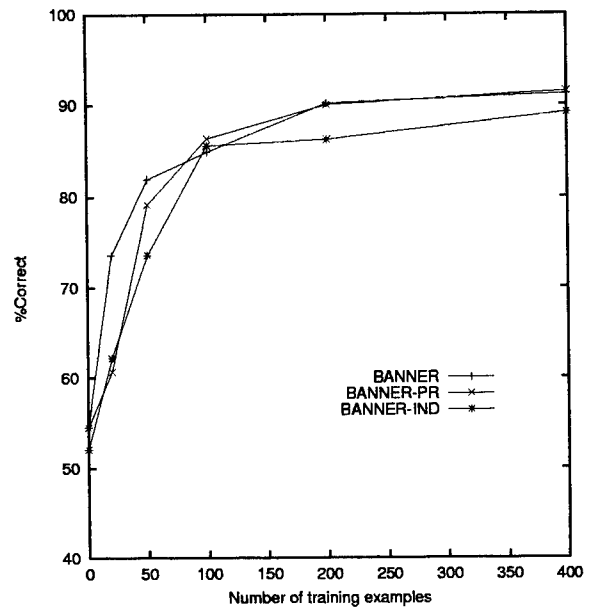


Figure 5: Splice-Junction: Banner Ablations

this problem. The data set contains 468 examples, consisting of strings of 57 nucleotides classified as promoters or non-promoters. Although in refinement experiments theories are sometimes corrupted randomly (Pazzani & Brunk, 1993), we found that the redundancy in this theory makes it very robust to small corruptions. Therefore, we generated a corrupt theory by deleting a portion of the theory we knew to be critical, namely the intermediate concept *minus_35* (deleted portion shown in bold in Figure 6).

Figure 7 shows BANNER-PR and BANNER's performance with this damaged theory compared to BAN-

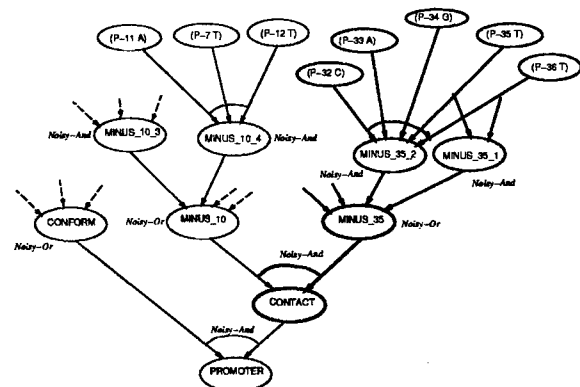


Figure 6: DNA Promoter Recognition - Initial Bayesian Network

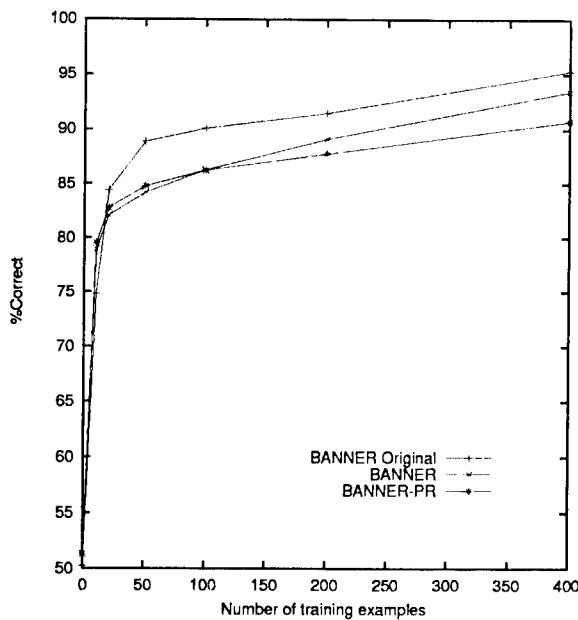


Figure 7: Effect of structure revision on corrupted promoter theory

NER's performance with the original theory. The graph shows that removing *minus_35* degrades the theory to the extent that, for most points in the learning curve, parameter revision alone cannot recover the accuracy attained with the original theory. The results shows that, for larger training sets, structure revision is effective at recovering a fair bit of the accuracy lost due to the corruption, although the difference between BANNER-PR and BANNER is only significant (at the 0.05 level) at 400 examples.

The fact that BANNER and BANNER-PR result in comparable accuracies for smaller training sets can be explained by the fact that none of the trials with 10 and 20 training examples, and less than half the trials with 50 examples required structure revision. Notice that the corrupted theory results in better networks than the original when trained on 10 examples. With 20 and 50 examples, the corrupted theory is still usually able to fit the training examples without structure revision, but results in poorer generalization. This leads to the hypothesis that, for smaller training sets, there are several theories that are as good as the original theory in fitting the training set, but are worse in terms of generalization, which would partially explain the observation that structure revision leads to improved training accuracies without any improvements in generalization, when trained on 50 and 100 examples.

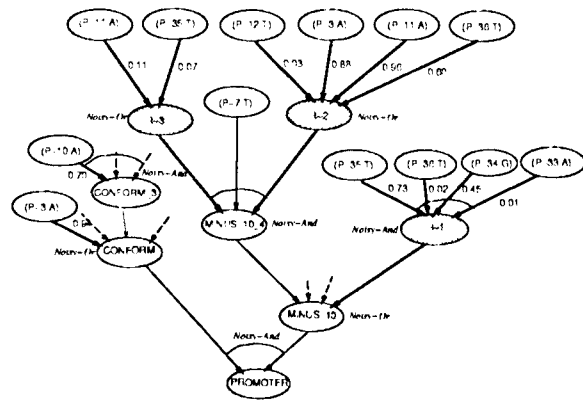


Figure 8: Example of a revised promoter network

Figure 8 illustrates a revised network. The nodes and links added by BANNER are indicated by shaded ellipses and bolder arrows and the numbers beside the links represent parameter values. Note that some nodes have been replicated in the figure for clarity only. BANNER added several features to the network: $P-35=T$, $P-36=T$, $P-34=G$, $P-33=A$ and $P-3=A$ and added new links from features already present in the network: $P-11=A$, and $P-10=A$. In addition, it has added three hidden variables, $I-1$ through $I-3$. A comparison with the original theory indicates that the added unit $I-1$ roughly corresponds to the deleted *minus_35* concept. However, in the original theory, *minus_35* combines conjunctively with *minus_10*, whereas, here it combines disjunctively. That could explain why BANNER also added some of these features to the sub-network above *minus_10.4*. However, realize that the initial theory is not known to have the correct structure, it is simply one proposed in the biological literature that is also consistent with the available data. Also, note that the modifications to the network are not confined to any particular level (as they are in Mahoney and Mooney (1994)).

In summary, our experiments demonstrate that BANNER is effective in revising an Bayesian networks with hidden variables to significantly improve their accuracy. They also demonstrate that the structure revision algorithm contributes significantly to the overall algorithm and makes semantically interpretable revisions. The effectiveness of the structure revision algorithm is also illustrated by the fact that BANNER-IND learns highly accurate classifiers. Experiments have also been performed that show that BANNER-IND learns more accurate classifiers than Naive Bayes on the problem of classifying chess end-games (Quinlan, 1983). Ramachandran (1998) provides details on

these results.

4 Related Work

While recent techniques have begun to address the problem of learning the structure of a Bayesian network from incomplete data (Ramoni & Sebastiani, 1997; Friedman, 1997), only a few address the problem of learning or revising networks with hidden variables. MS-EM (Friedman, 1997) extends EM to learn the structure as well as the parameters of a network from incomplete data. While it works when the initial theory contains hidden variables, it cannot construct new hidden variables. Kwoh and Gillies (1996) present a procedure for adding hidden variables by first learning a Bayesian network from data without hidden variables, and then using statistical analysis to find correlations between variables with the same cause and clustering such variables with a new hidden node. These techniques have been demonstrated on learning small networks, but have not been evaluated on larger, real-world problems. Moreover, it has no mechanism for selecting a candidate set of nodes that need to be revised, instead relying on blind search through the space of all possible revisions.

5 Future Research

Experiments on other realistic problems, particularly ones in which the initial theory is specified as a Bayesian network (rather than translated from rules), is one area for future research. The current results for BANNER involve problems of causal inference, tests on tasks involving abductive inference are also needed. More detailed comparisons of different Bayes-net induction and revision algorithms and competing methods on realistic problems measuring both training time and predictive accuracy are clearly needed. The current literature on Bayes-net learning is particularly lacking in this regard relative to other areas of machine learning (Friedman, Goldszmidt, Heckerman, & Russell, 1997).

Extending BANNER's general approach to handle nodes other than noisy-or/and ones is an important area for future study. Another is theory refinement for unsupervised learning where there is not a specific targeted inference task. The algorithm can also be extended to use Bayesian metrics to select new nodes to be added to the parent set of a node. A number of interesting ideas for learning and revising Bayes nets have been proposed, but integrating them into an ef-

ficient and effective system with clearly demonstrated advantages over other machine-learning methods on realistic problems is still a challenge.

6 Conclusion

We have introduced a novel technique for revising Bayesian networks that can handle existing hidden variables as well as create new ones. We have demonstrated, through experiments on realistic problems, that this approach can efficiently revise large networks and produce highly accurate classifiers. The results are also competitive with those of the best theory refinement systems while maintaining the precise probabilistic semantics of Bayesian networks that we believe make the resulting theories significantly more comprehensible. Whereas existing techniques for revising Bayesian networks must search through the space of all possible revisions, we have presented novel mechanisms for using the information in the data to guide the search for useful revisions, thus focusing the search and making it tractable for larger, more realistic problems.

7 Acknowledgements

We are grateful to Bobby Blumofe, Lorenzo Alvisi, Mike Dahlin, Calvin Lin and other folks at the UT LESS lab for letting us use their computing facilities for this research. The multiprocessor computing facilities in this lab were made available through a generous equipment donation from Sun Microsystems. This research was partially supported by the National Science Foundation through grants IRI-9310819 and IRI-9704943.

References

- Baffes, P. T., & Mooney, R. J. (1996). A novel application of theory refinement to student modeling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 403-408 Portland, OR.
- Brunk, C., & Pazzani, M. (1995). A lexically based semantic bias for theory revision. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 81-89 San Francisco, CA. Morgan Kaufman.
- Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 52-60.
- Friedman, N., Goldszmidt, M., Heckerman, D., & Russell, S. (1997). Challenge: What is the impact of Bayesian networks on learning?.. pp. 10-15 Nagoya, Japan.

- Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 125-133 Nashville, Tennessee. Morgan Kaufmann Publishers.
- Friedman, N., & Goldszmidt, M. (1996). Building classifiers using Bayesian networks. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1277-1284.
- Heckerman, D. (1995). A tutorial on learning Bayesian networks. Tech. rep. MSR-TR-95-06, Microsoft Research, Redmond, WA.
- Kohavi, R., Becker, B., & Sommerfield, D. (1997). Improving simple Bayes. In *Proceedings of the European Conference on Machine Learning*.
- Kwoh, C.-K., & Gillies, D. (1996). Using hidden nodes in Bayesian networks. *Artificial Intelligence*, 88(1-2), 1-38.
- Lam, W., & Bacchus, F. (1994). Using new data to refine a Bayesian network. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 383-390.
- Mahoney, J. J., & Mooney, R. J. (1994). Comparing methods for refining certainty-factor rule bases. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 173-180 New Brunswick, NJ.
- McClelland, J. L., & Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. The MIT Press, Cambridge, MA.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York, NY.
- Mooney, R. J. (1997). Integrating abduction and induction in machine learning. In *Working Notes of the IJCAI-97 Workshop on Abduction and Induction in AI*, pp. 37-42 Nagoya, Japan.
- Opitz, D. W., & Shavlik, J. W. (1993). Heuristically expanding knowledge-based neural networks. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 512-517 Chambray, France.
- Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66, 311-344.
- Pazzani, M., & Brunk, C. (1993). Finding accurate frontiers: A knowledge-intensive approach to relational learning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 328-334 Washington, D.C.
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Inc., San Mateo, CA.
- Pradhan, M., Provan, G., Middleton, B., & Henrion, M. (1994). Knowledge engineering for large belief networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 484-490 Seattle, WA.
- Provan, G. M., & Singh, M. (1994). Learning Bayesian networks using feature selection. In *Proceedings of the Workshop on Artificial Intelligence and Statistics*, pp. 291-300 New York. Springer-Verlag.
- Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, Los Altos, CA.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81-106.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.
- Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3), 239-266.
- Ramachandran, S. (1998). *Theory Refinement of Bayesian Networks with Hidden Variables*. Ph.D. thesis, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 98-265 (see <http://www.cs.utexas.edu/users/ai-lab>).
- Ramachandran, S., & Mooney, R. J. (1996). Revising Bayesian networks parameters using backpropagation. In *International Conference on Neural Networks: Plenary, Panel and Special Sessions*, pp. 82-87 Washington D.C., USA.
- Ramoni, M., & Sebastiani, P. (1997). Learning Bayesian networks from incomplete databases. In Geiger, D., & Shenoy, P. (Eds.), *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, Inc.
- Russell, S., Binder, J., Koller, D., & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 1146-1152 Montreal, Canada.
- Thiesson, B. (1995). Accelerated quantification of Bayesian networks with incomplete data. In Fayyad, U. M., & Uthurusamy, R. (Eds.), *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 306-11. AAAI Press.
- Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70, 119-165.

Learning to Drive a Bicycle using Reinforcement Learning and Shaping

Jette Randløv

CATS, Niels Bohr Institute,
University of Copenhagen,
Blegdamsvej 17,
DK-2100 Copenhagen Ø, Denmark
randlov@nbi.dk

Preben Alstrøm,

alstrom@cats.nbi.dk

Abstract

We present and solve a real-world problem of learning to drive a bicycle. We solve the problem by online reinforcement learning using the Sarsa(λ)-algorithm. Then we solve the composite problem of learning to balance a bicycle and then drive to a goal. In our approach the reinforcement function is independent of the task the agent tries to learn to solve.

1 Introduction

Here we consider the problem of learning to balance on a bicycle. Having done this we want to drive the bicycle to a goal. The second problem is not as straightforward as it may seem. The learning agent has to solve two problems at the same time: Balancing on the bicycle and driving to a specific place. Recently, ideas from behavioural psychology have been adapted by reinforcement learning to solve this type of problem. We will return to this in section 3.

In reinforcement learning an agent interacts with an environment or a system. At each time step the agent receives information on the state of the system and chooses an action to perform. Once in a while, the agent receives a reinforcement signal r . Receiving a signal could be a rare event or it could happen at every time step. No evaluative feedback from the system other than the failure signal is available. The goal of the agent is to learn a mapping from states to actions that maximizes the agent's discounted reward over time [Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 1998]. The discounted reward is the sum $\sum_{i=0}^{\infty} \gamma^i r_{t+i}$, where γ is the discount parameter.

A lot of techniques have been developed to find near optimal mappings on a trial-and-error basis. In this paper we use the Sarsa(λ)-algorithm, developed by Rummery and

1. Initialize all eligibility traces $e_0 = 0$.
2. Set $t = 0$.
3. Choose action a_t .
4. If $t > 0$ then learn
 $w_t = w_{t-1} + \alpha [r_{t-1} + \gamma Q_t - Q_{t-1}] e_{t-1}$.
5. Calculate $\nabla_w Q_t$ with respect to the chosen action.
6. Update accumulating traces as
 $e_t = \gamma \lambda e_{t-1} + \nabla_w Q_t$.
 Update replacing traces as

$$e_t(s) = \begin{cases} \nabla_w Q_t & \text{if } \nabla_w Q_t \neq 0, \\ \gamma \lambda e_{t-1}(s) & \text{otherwise.} \end{cases}$$
7. Perform action, receive reinforcement-signal.
8. If the system has entered a terminal state, then $t \leftarrow t + 1$ and jump to point 3.
9. Otherwise perform the learning (point 4) with $Q_t = 0$.

Figure 1: The Sarsa(λ)-algorithm.

Niranjan [Rummery and Niranjan, 1994, Rummery, 1995, Singh and Sutton, 1996, Sutton and Barto, 1998], because empirical studies seem to suggest that this algorithm is the best so far [Rummery and Niranjan, 1994, Rummery, 1995, Sutton and Barto, 1998]. Figure 1 shows the Sarsa(λ)-algorithm. We have modified the algorithm slightly by cutting of eligibility traces that fall below 10^{-7} in order to save calculation time. For replacing traces we allowed the trace for each state-action pair to continue until that pair occurred again, contrary to Singh and Sutton [Singh and Sutton, 1996].

2 Learning to balance on a bicycle

Our first task is to learn to balance. At each time step the agent receives information about the state of the bicycle,

the angle and angular velocity of the handle bars, the angle, angular velocity and acceleration of the angle from the bicycle to vertical. For details of the bicycle system we refer to appendix A.

The agent chooses two basic actions. What torque should be applied to the handle bars, $T \in \{-2\text{ N}, 0\text{ N}, +2\text{ N}\}$, and how much the centre of mass should be displaced from the bicycle's plan, $d \in \{-2\text{ cm}, 0\text{ cm}, +2\text{ cm}\}$ — a total of 9 possible actions. Noise is laid on the choice of displacement, to simulate an imperfect balance, $d = d_{\text{agents choice}} + sp$, where p is a random number within $[-1; 1]$ and s is the noise level measured in centimeters. We use $s = 2\text{ cm}$.

Our agent consists of 3456 input neurons and 9 output neurons, with full connectivity and no hidden layers. The learning rate is $\alpha = 0.5$. The continuous state data is discretised by non-overlapping intervals in the state-space, such that there is exactly one active neuron in the input layer. This neuron represent state information for all the different state variables. The discrete intervals (boxes) are based on the following quantization thresholds:

The angle the handle bars are displaced from normal, θ : $0, \pm 0.2, \pm 1, \pm \frac{\pi}{2}$ radians.

The angular velocity of the angle, $\dot{\theta}$: $0, \pm 2, \pm \infty$ radians/second.

The angle from vertical to bicycle, ω : $0, \pm 0.06, \pm 0.15, \pm \frac{1}{15}\pi$ radians.

The angular velocity, $\dot{\omega}$: $0, \pm 0.25, \pm 0.5, \pm \infty$ radians/second.

The angular acceleration, $\ddot{\omega}$: $0, \pm 2, \pm \infty$ radians/second².

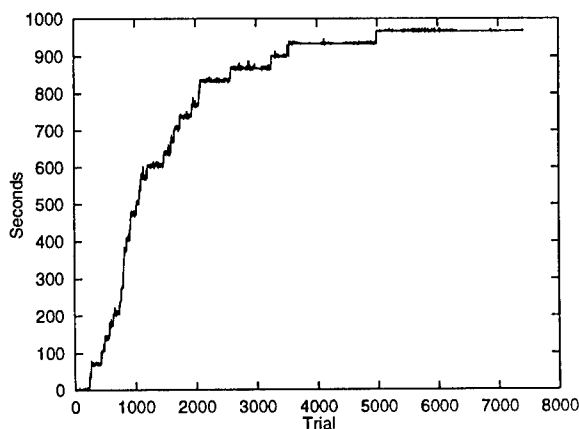


Figure 2: Number of seconds the agent can balance on the bicycle, as a function of the number of trials. Average of 40 agents. (After the agent has learned the task, 1000 seconds are used in calculation of the average.)

Figure 2 shows the number of seconds the agent can balance on the bicycle as a function of the number of trials. When the agent can balance for 1000 seconds, the task is considered learned. Here $\lambda = 0.95$ and $\gamma = 0.99$. Several CMAC-systems (also known as generalized grid coding) [Watkins, 1989, Santamaría et al., 1996, Sutton, 1996, Sutton and Barto, 1998], were also tried, but none of them gave the agent a learning time below 5000 trials.

Figures 3 and 4 show the movements of the bicycle at the beginning of a learning process seen from above. Each time the bicycle falls over it is restarted at the starting point. At each time step a line is drawn between the points where the tyres touch the ground.

Both accumulating and replacing eligibility traces were tried. The results are shown in figure 5. The results found support the general conclusions drawn by Singh and Sutton [Singh and Sutton, 1996]: Replacing traces make the agent perform much better than conventional, accumulating traces. Long traces help the agent best.



Figure 3: The first 151 trials seen from above. The longest path is 7 meters.

3 Shaping

The idea of shaping, which is borrowed from behavioural psychology, is to give the learning agent a series of relatively easy problems building up to the harder problem of ultimate interest [Sutton and Barto, 1998]. The term originates from the psychologist Skinner [Skinner, 1938], who studied the effect on animals, especially pigeons and rats.

To train an animal to produce a certain behavior, the trainer must find out what subtasks constitute an approximation of the desired behavior, and how these should be reinforced [Staddon, 1983]. By rewarding successive approximations to the desired behavior, pigeons can be brought to pecking a selected spot [Skinner, 1953, p. 93], horses to do clever tricks in a circus like seemingly recognize flags of nations or numbers and to do calculation [Jørgensen, 1962, pp. 137-139], and pigs to perform complex acts as eating breakfast at a table and vacuuming the floor [Atkinson et al., 1996, p. 242]. Staddon notes that human education as well is built up as a process of shaping if behavior is taken to include "understanding" [Staddon, 1983, p. 458].



Figure 4: The same route as figure 3 a little later. Now the agent can balance the bicycle for 30–40 meters. The agent starts each trial in a equilibrium position $(\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}) = (0, 0, 0, 0, 0)$. During the first trials it learns to avoid disturbing this unnecessarily, i.e. it learns to keep driving straight forward. Now the most difficult part of the learning remains: To learn to come safe though a dangerous situation. A weak (random) preference for turning right (instead of left) is strengthened during the learning as the agent gets better at handling problematic situations and therefore receives less discounted punishment than expected.

Shaping can be used to speed up the learning process for a problem or in general to help the reinforcement learning technique scale to large and more complex problems. But there is a price to be paid for faster learning: We must give up the *tabula rasa* attitude that is one of the attractive aspects of basic reinforcement learning. To use shaping in practice one must know more about the problem than just under which conditions an absolute good or bad state has been reached. This introduces the risk that the agent learns a solution to a problem that is only locally optimal.

There are at least three ways to implement shaping in reinforcement learning: By lumping basic actions together as macro-actions, by designing a reinforcement function that rewards the agent for making approximations to the desired behavior, and by structurally developing a multi-level architecture that is trained part by part.

Selfridge, Sutton and Barto showed that transferring knowledge from solving an easy version of a problem such as the classical pole mounted on a cart can ease learning a more difficult version [Selfridge et al., 1985].

McGovern, Sutton and Fagg have tested macro-actions in a gridworld and found that in some cases they accelerate the learning process [McGovern et al., 1997].

Dorigo, Colombetti and Borghi have worked with shaping for real robots [Dorigo and Colombetti, 1993, Colombetti et al., 1996, Dorigo and Colombetti, 1997]. They use reinforcement learning as a mean to translate suggestions from an external trainer. The trainer is a programme in itself with a high-level representation of the desired behavior that provided immediate reinforcement. For instance in the “The Hamster Experiment” [Colombetti et al., 1996] the robot’s task is to collect pieces of food (colored cans) and bring them to its nest. The trainer provides the agent with a reinforcement signal for approaching the food. This signal is proportional to the decrease in the distance between the robot and the pieces of food. The training of the agent boils down to translating the high-level trainer to a low-level control programme. This method of shaping by a trainer has a number of advantages as well as disadvantages. The agent does not have to solve the delayed reinforcement problem. But on the other hand, the programmer of the trainer must know in advance what high-level behavior is desired, and to such a degree that the trainer can judge how well a single move fits into the desired behavior.

Mataric has studied the possibility of putting implicit domain knowledge into the agent by construction a more complex reinforcement function than commonly used [Mataric, 1994]. Again the theory was tested on a real robot moving cans to a nest. Here the constructed function did

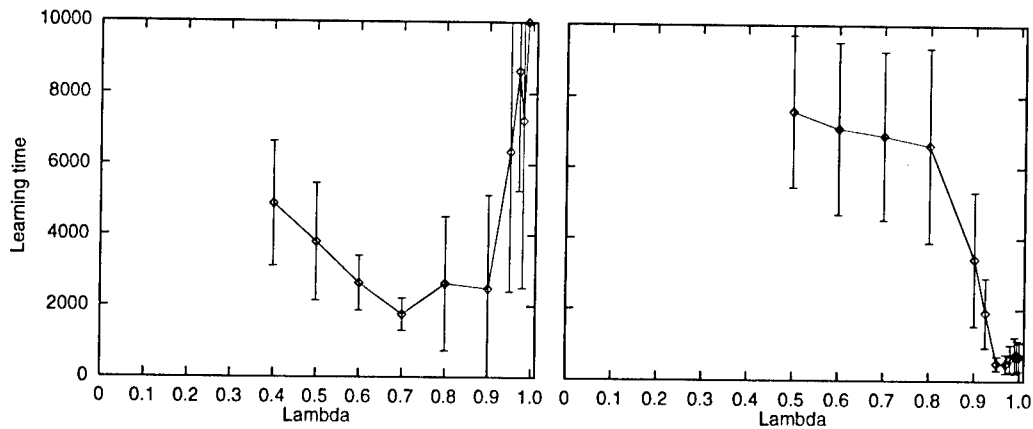


Figure 5: Learning time for different values of λ for accumulating eligibility traces (left) and replacing traces (right). Each point is an average of 30 simulations.

not eliminate the need for solving the delayed reinforcement problem.

Gullapalli has studied two implementations of shaping [Gullapalli, 1992]. In the first the complexity of the control task is gradually increased during learning, and the reinforcement function used is changed accordingly. In this way most of a training run is used in learning the approximation to the current target behavior. This system was used to make a simulated robot hand perform a series of key strokes on a calculator. The actual task consisted of six subtasks. Secondly Gullapalli considered structural shaping: An incremental development of the learning system where a multi-level architecture is trained in parts.

Gerald Tesauro's Backgammon playing agent achieved master level play through self-play [Tesauro, 1992, Tesauro, 1994, Tesauro, 1995]. This can be considered as a very successful example of the use of shaping. Self-play is a sort of shaping, since at first the agent plays against a nearly random opponent and thereby solves an easy task. The complexity of the task then grows as the agent gets better at playing.

In Gullapalli's experiments [Gullapalli, 1992] and Selfridge, Sutton and Barto's [Selfridge et al., 1985], as well as in Dorigo, Colombetti and Borghi's [Colombetti et al., 1996, Dorigo and Colombetti, 1997], the agent received a different reinforcement signal over time for the same behavior. This is not in agreement with the original inspiration of the reinforcement signal as being a hardwired signal inside the brain of a animal. To solve this problem, we need the reinforcement function to be independent of what task the agent tries to learn to solve. Our approach in general is to let the most basic tasks result in the lowest reinforcement signals and more advanced tasks correspond to larger signals.

Say, we want a robot to learn to move forward like a child (see figure 6). As a child grows stronger it discovers more complex and faster ways of moving. Performing each way of moving can be seen as a task that is more difficult than the former. The robot starts by learning to roll.

Having done so, it might discover how to crawl. The reinforcement signal for crawling is greater than rolling, and greater than what the agent expects to receive, and therefore it acts as a reward. Later after having learned to walk, failing to walk and falling back on crawling makes the robot receive a smaller reinforcement signal than it expected, and the internal reinforcement signal becomes negative—that is the signal acts as a punishment.

Can these basic ideas of shaping be applied to reinforcement learning, and make it possible to solve a complex problem with more than one goal? We will now turn to a practical study of these theoretical issues.

4 Learning to drive to a goal using shaping

We want to study shaping on the composite problem of learning to balance a bicycle and then drive to a goal. In contrast to other experiments with shaping, we want the agent to be totally in charge of when to switch task. When

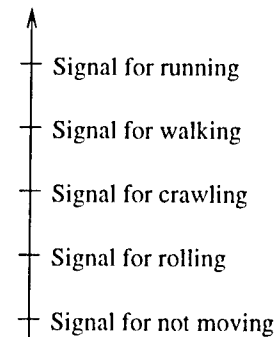


Figure 6: Reinforcement signals for the movements of a child-robot.

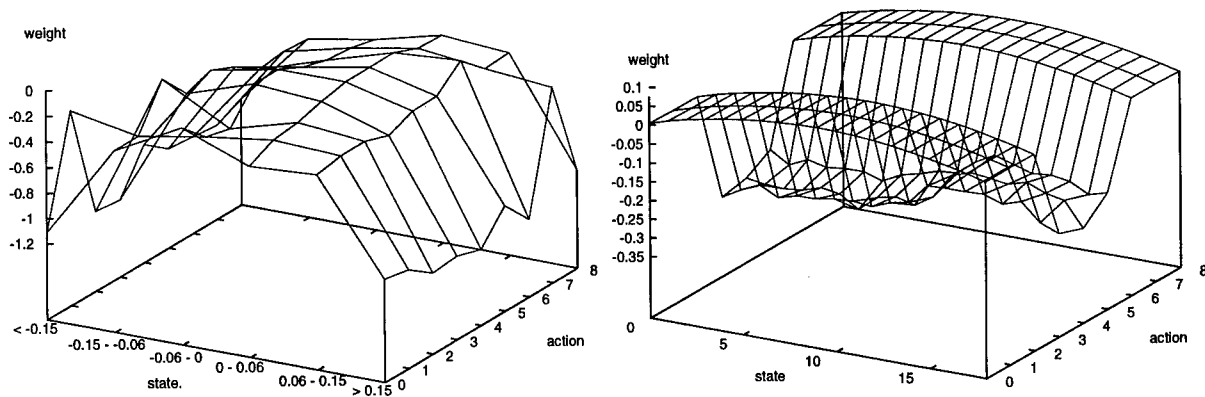


Figure 7: The weight, from the ω -oriented input neurons (left) and the weights from the angle oriented input neurons (right). Note the difference of the scale.

one drives a bicycle in the morning to the institute and hits a hole in the road, one instantaneously forgets about where to go and focus attention on the balance. We want the agent to be able to switch task equally swiftly when it find the situation appropriate.

The bicycle starts out at the origin heading west. The goal is a circular spot (10 meter radius) positioned 1000 meters to the north of the starting point.

We enlarge our basic network by 20 more input neurons, with full connectivity to the 9 output neurons. The angle between the driving direction and the direction to goal is discretised by 18° intervals, one for each neuron. Now there are exactly two active neurons in the agents input layer—one for the state of the bicycle and one for the driving direction relative to the goal. The learning rate for the weights from the angle-input neurons is chosen to be 0.01—much smaller than the rate for the other weights, in order to reflect the different time scales in the learning tasks: We do not want the weights in the angle oriented part to grow large while the agent learns to balance the bicycle. The odds are against these weights ending up containing anything useful.

The reinforcement function is independent of the task the agent tries to learn to solve. If the bicycle falls over, the agent always receives -1 , if the agent reaches the goal it is rewarded by $r = 0.01$, and otherwise the agent receives $r = (4 - \psi_g^2) \cdot 0.00004$, where ψ_g is the angle between the driving direction and the direction to goal measured in radians. The agent is punished when driving away from the goal and rewarded when driving towards it. This reinforcement function is inspired by the signal used by Colombetti, Dorigo and Borghi [Colombetti et al., 1996] mentioned earlier. Note that the agent still have to solve the delayed reinforcement problem. As one can see, the

numerical value of this signal is quite small. We tried larger values, which made the agent learn to drive in the correct orientation without being able to balance. After a few hundred trials the agent at the starting point immediately threw the bicycle to the right. The positive reinforcement it received due to the correct orientation in several time steps was large enough to make up for the punishment from falling.

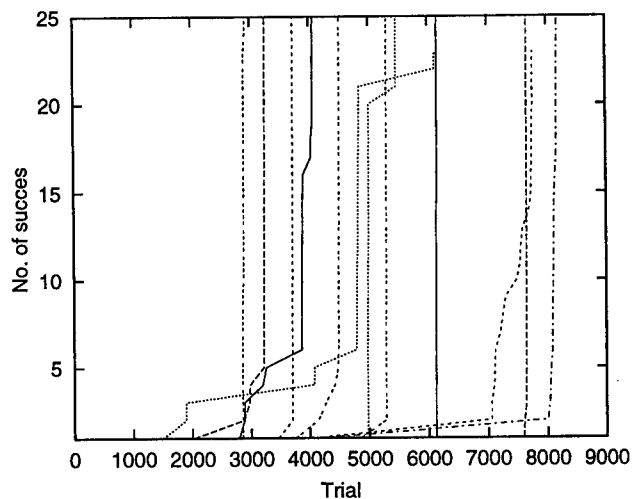


Figure 8: Number of times an agent drives the bicycle to the goal for twelve agents.

Figure 8 shows the number of times twelve agents reach the goal. In a typical learning process it takes the agent 1700 trials to learn to balance (i.e. drive more than 1000 s without falling), and after about 4200 trials it gets to the goal for the first time. After a total of approximately 5700 trials it drives to the goal more or less every time.

Figure 7 shows the values of some of the important weights

after learning. The ω -weights shown are an average of weight values around $\theta = 0$, $\dot{\theta} = 0$, $\dot{\omega} = 0$ and $\ddot{\omega} = 0$. If the agent drives along in balance, the weights with values in the relatively flat upper area are active for the balance oriented input neurons, and the values of the angle oriented neurons matter for the choice of action. The weights belonging to the balance oriented input neurons makes the agent prefer action 3, 4 and 5 (which corresponds to $T = 0$), but the weights belonging to the angle oriented neurons decide which one. But if the state of the bicycle enters an area of unbalance, the balance oriented input neurons have far greater differences in values of the weights, and as a result the angle oriented input neurons do not make any difference for the choice of action. In other words: The agent swiftly shifts attention from the task of finding the goal to the task of balancing the bicycle if required.



Figure 9: A typical route when the agent reaches the goal for the first time.

Figure 9 and 10 shows routes from the starting point to the goal (the grey circle on the y-axis). The first drives to the goal can be as long as 200 km, but the agent soon learns to drive to the goal driving “only” 7 km. A driving distance as short as 1680 m has been observed.

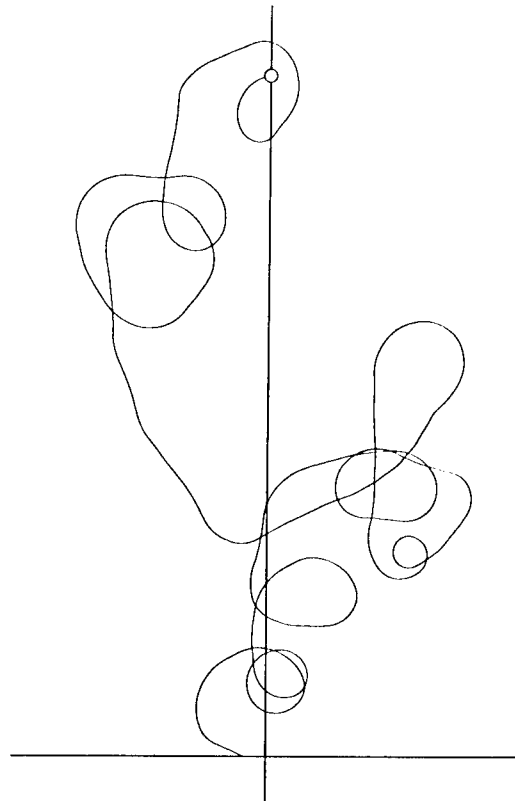


Figure 10: Already after 10 drives to the goal the agent navigates a little better.

The goal is not reached just by coincidence. The probability for hitting the goal at random is quite small. An estimate for the time required to reach the goal by doing a correlated random walk is 10^{10} time steps. (The bee line from the starting point to the goal is $3.6 \cdot 10^4$ time steps.) In other words: If the agent had to solve the problem of learning to drive to the goal without access to the shaping reinforcement signal, i.e. the tabular rasa approach, it would take enormous amounts of time before it hits the goal for the first time and experiences the reward for getting there.

We agree with Mataric [Mataric, 1994] that these heterogeneous reinforcement functions have to be designed with great care. In our first experiments we rewarded the agent for driving towards the goal but did not punish it for driving away from it. Consequently the agent drove in circles with a radius of 20–50 meters around the starting point. Such behavior was actually rewarded by the reinforcement function, furthermore circles with a certain radius are physically very stable when driving a bicycle because of the cross terms in eqs. (2) and (3) in the appendix.

5 Conclusion

Our results demonstrate the utility of reinforcement learning on a difficult, dynamical real world problem. It is possible to learn to balance a bicycle by pure reinforcement learning with only one (rare) reinforcement signal. Furthermore it is possible to learn a solution to the double problem of balancing on the bicycle and driving to a goal by combining reinforcement learning with shaping. The application of shaping accelerated the learning process immensely. Without shaping, it would not have been practical to wait for the agent to discover the goal and the reward for getting there.

Acknowledgements

We would very much like to thank Andrew G. Barto for several good discussions and stimulating ideas.

References

- [Atkinson et al., 1996] Atkinson, R. L., Atkinson, R. C., Smith, E. E., Bem, D. J., and Nolen-Hoeksema, S. (1996). *Hilgard's Introduction to Psychology*. Harcourt Brace College Publishers, 12th edition.
- [Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.
- [Colombetti et al., 1996] Colombetti, M., Dorigo, M., and Borghi, G. (1996). Robot shaping: The hamster experiment. Technical Report TR/IRIDIA/1996-6, Université Libre de Bruxelles.
- [Dorigo and Colombetti, 1993] Dorigo, M. and Colombetti, M. (1993). Robot shaping: Developing autonomous agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley. Labeled: To appear in Artificial Intelligence Journal.
- [Dorigo and Colombetti, 1997] Dorigo, M. and Colombetti, M. (1997). Précis of "Robot Shaping: An Experiment in Behavior Engineering". *Adaptive Behavior*, 5(3-4). Précis of the book from MIT Press, Oct. 1997.
- [Gullapalli, 1992] Gullapalli, V. (1992). *Reinforcement Learning and Its Application to Control*. PhD thesis, University of Massachusetts. COINS Technical Report 92-10.
- [Jørgensen, 1962] Jørgensen, J. (1962). *Psykologi – paa biologisk Grundlag*. Scandinavian University Books. Munksgaard, København.
- [Mataric, 1994] Mataric, M. J. (1994). Reward functions for accelerated learning. In Cohen, W. W. and Hirsh, H., editors, *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann, CA.
- [McGovern et al., 1997] McGovern, A., Sutton, R. S., and Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. In 1997 *Grace Hopper Celebration of Women in Computing*.
- [Rummery, 1995] Rummery, G. A. (1995). *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University Engineering Department.
- [Rummery and Niranjan, 1994] Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University.
- [Santamaría et al., 1996] Santamaría, J. C., Sutton, R. S., and Ram, A. (1996). Experiments with reinforcement learning in problems with continuous states and action spaces. Technical Report 96-088, COINS.
- [Selfridge et al., 1985] Selfridge, O. G., Sutton, R. S., and Barto, A. G. (1985). Training and tracking in robotics. In *Proceedings of the Ninth International Joint Conference in Artificial Intelligence*, pages 670-672. Morgan Kaufmann, CA.
- [Singh and Sutton, 1996] Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123-158.
- [Skinner, 1938] Skinner, B. F. (1938). *The Behavior of Organisms: An Experimental Analysis*. Prentice Hall, Englewood Cliffs, New Jersey.
- [Skinner, 1953] Skinner, B. F. (1953). *Science and Human Behavior*. Collier-Macmillan, New York.
- [Staddon, 1983] Staddon, J. E. R. (1983). *Adaptive Behavior and Learning*. Cambridge University Press.
- [Sutton, 1996] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038-1044. The MIT Press, Cambridge.
- [Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press/Bradford Books.
- [Tesauro, 1992] Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257-277.
- [Tesauro, 1994] Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. Technical report, IBM, Thomas J. Watson Research Center, Yorktown Heights, NY 10598.
- [Tesauro, 1995] Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38.
- [Watkins, 1989] Watkins, C. J. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University.

A Details of the Bicycle Simulation

The bicycle must be held upright within $\pm 12^\circ$ measured from vertical position. If the angle from the vertical to the bicycle falls outside this interval, the bicycle has fallen, and

the agent receives punishment -1 . The Bicycle is modeled by the following non-linear differential equations. One simplification was made to ease the derivation of the equations: The front fork was assumed to be vertical, which is unusual but not impossible. This, however, made the task a bit more difficult for the agent.

There are two important angles in this problem: The angle θ of the direction of the bicycle from straightforward, and the angle ω the bicycle is tilted from vertical. The conservations of angular momentum of the tyres results in some important cross terms.

The equations do not model a bicycle exactly, as some second order cross effects were ignored during the derivation. However we believe that the largest problem of transferring to a real bicycle would be to build hardware that could withstand falling over a thousand times—not just without crashing but also without changing and thereby make the system unstationary.

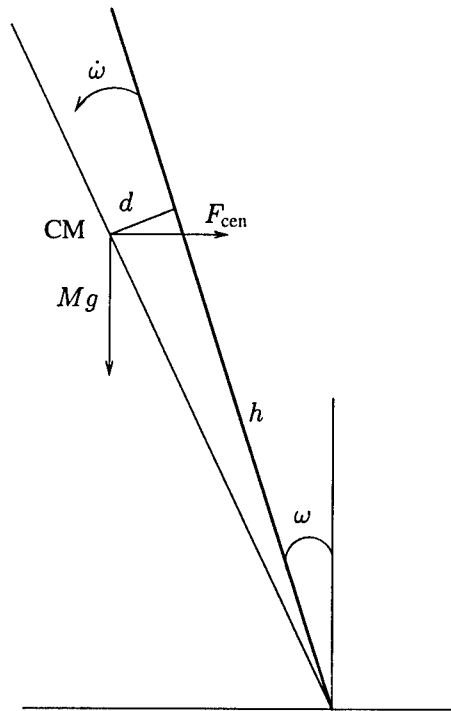


Figure 11: The bicycle as seen from behind. The thick line represents the bicycle. CM is the centre of mass of the bicycle and cyclist.

The following equations describe the mechanics of the system. (See figure 11.) The angle φ is the total angle of tilt of the centre of mass, and is defined as:

$$\varphi \stackrel{\text{def}}{=} \omega + \arctan\left(\frac{d}{h}\right) \quad (1)$$

The angular acceleration $\ddot{\omega}$ can be calculated as:

$$\ddot{\omega} = \frac{1}{I_{\text{bicycle and cyclist}}} \left(Mhg \sin \varphi - \cos \varphi \left(I_{dc} \dot{\sigma} \dot{\theta} + \text{sign}(\theta) \cdot v^2 \left(\frac{M_d r}{r_f} + \frac{M_d r}{r_b} + \frac{Mh}{r_{CM}} \right) \right) \right) \quad (2)$$

This equation is the mechanical equation for angular momentum. The physical contents of the right hand side are terms for the gravitation, effects of the the conservation of angular momentum of the tyres and the fictional centrifugal force. The term $I_{dc} \dot{\sigma} \dot{\theta}$ is important for understanding why it is relative easier to ride a bicycle than to keep the balance on a bicycle standing still. The cross effects that originate from the conservation of angular momentum of the tyres stabilize the bicycle, and this effect is proportional to the angular velocity of the tyres $\dot{\sigma}$ and thereby to the velocity of the bicycle.

The angular acceleration $\ddot{\theta}$ of the front tyre and the handle bars is:

$$\ddot{\theta} = \frac{T - I_{dv} \dot{\sigma} \dot{\omega}}{I_{dt}} \quad (3)$$

These equations are not an exact analytical description, as some second (and higher) order terms have been ignored. The values of ω , $\dot{\omega}$, $\ddot{\omega}$, θ , $\dot{\theta}$ are send to the agent at each time step. The agent returns the value of d and the torque T .

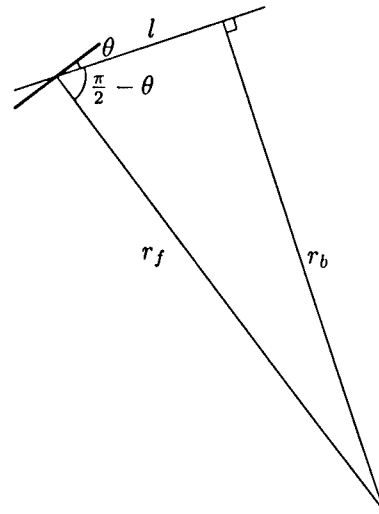


Figure 12: Seen from above. The thick line represents the front tyre.

The front and back tyres follow different paths in a curve with different radii (see figure 12). The front tyre follows

the longest path. The radius for the front tyre is:

$$r_f = \frac{l}{|\cos(\frac{\pi}{2} - \theta)|} = \frac{l}{|\sin \theta|} \quad (4)$$

And for the back tyre:

$$r_b = l \left| \tan \left(\frac{\pi}{2} - \theta \right) \right| = \frac{l}{|\tan \theta|} \quad (5)$$

For the CM the radius can be calculated as:

$$r_{CM} = \left((l - c)^2 + \frac{l^2}{(\tan \theta)^2} \right)^{\frac{1}{2}} \quad (6)$$

The equations of the position of the tyres for the front tyre:

$$\begin{pmatrix} x_f \\ y_f \end{pmatrix}_{(t+1)} = \begin{pmatrix} x_f \\ y_f \end{pmatrix}_{(t)} + v dt \begin{pmatrix} -\sin(\psi + \theta + \text{sign}(\psi + \theta) \arcsin(\frac{v dt}{2r_f})) \\ \cos(\psi + \theta + \text{sign}(\psi + \theta) \arcsin(\frac{v dt}{2r_f})) \end{pmatrix}$$

And for the back tyre:

$$\begin{pmatrix} x_b \\ y_b \end{pmatrix}_{(t+1)} = \begin{pmatrix} x_b \\ y_b \end{pmatrix}_{(t)} + v dt \begin{pmatrix} -\sin(\psi + \text{sign}(\psi) \arcsin(\frac{v dt}{2r_b})) \\ \cos(\psi + \text{sign}(\psi) \arcsin(\frac{v dt}{2r_b})) \end{pmatrix}$$

We estimated the values of the moments of inertia to:

$$I_{\text{bicycle and cyclist}} = \frac{13}{3} M_c h^2 + M_p (h + d_{CM})^2 \quad (7)$$

The various moments of inertia for a tyre was estimated to (see figure 13):

$$I_{dc} = M_d r^2 \quad (8)$$

$$I_{dv} = \frac{3}{2} M_d r^2 \quad (9)$$

$$I_{dl} = \frac{1}{2} M_d r^2 \quad (10)$$

Table 1 shows the values of the parameters used for the bicycle system.

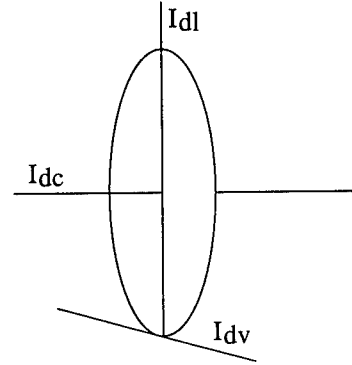


Figure 13: Axis for moments of inertia for a tyre.

Notation		Value
c	Horizontal distance between the point, where the front wheel touches the ground and the CM.	66 cm
CM	The Centre of Mass of the bicycle and cyclist as a total	
d	The agent's choice of the displacement of the CM perpendicular to the plan of the bicycle	
d_{CM}	The vertical distance between the CM for the bicycle and for the cyclist.	30 cm
h	Height of the CM over the ground	94 cm
l	Distance between the front tyre and the back tyre at the point where they touch the ground	111 cm
M_c	Mass of the bicycle	15 kg
M_d	Mass of a tyre	1.7 kg
M_p	Mass of the cyclist	60 kg
r	Radius of a tyre	34 cm
$\dot{\sigma}$	The angular velocity of a tyre	$\dot{\sigma} = \frac{v}{r}$
T	The torque the agent applies on the handlebars	
v	The velocity of the bicycle	10 km/h

Table 1: Notation and values for the bicycle system.

Learning First-Order Acyclic Horn Programs from Entailment*

Chandra Reddy Prasad Tadepalli

Dearborn 303

Department of Computer Science

Oregon State University,

Corvallis, OR 97331-3202.

{reddyc,tadepalli}@cs.orst.edu

Abstract

In this paper, we consider learning first-order Horn programs from entailment. In particular, we show that any subclass of first-order acyclic Horn programs with constant arity is exactly learnable from equivalence and entailment membership queries provided it allows a polynomial-time subsumption procedure and satisfies some closure conditions. One consequence of this is that first-order acyclic determinate Horn programs with constant arity are exactly learnable from equivalence and entailment membership queries.

1 Introduction

Learning first-order Horn programs—sets of first-order Horn clauses—is an important problem in inductive logic programming with applications ranging from speedup learning to grammatical inference.

We are interested in speedup learning, which concerns learning domain-specific control knowledge to alleviate the computational hardness of planning. One kind of control knowledge, which is particularly useful in many domains, is represented as goal-decomposition rules. Each decomposition rule specifies how a goal can be decomposed into a sequence of subgoals, given that a set of conditions is true in the initial problem state. Each of the subgoals might in turn have a set of decomposition rules, unless it is a primitive action, in which case it can be directly executed.

Unlike in logical inference, for which Horn clauses are ideally suited, in planning, one needs to keep track of

time. In spite of this difference, goal-decomposition rules can be represented as first-order Horn clauses by adding two situation variables to each literal to indicate the time interval in which the literal is true. Hence, the problem of learning goal-decomposition rules for a *single* goal can be mapped to learning first-order Horn definitions—a set of Horn clauses, all having the same head or consequent literal. Learning goal-decomposition rules for *multiple* goals corresponds to learning first-order Horn programs. Henceforth, we omit the prefix “first-order”, except when there is a possibility of ambiguity.

In learning from entailment, a positive (negative) example is a Horn clause that is implied (not implied) by the target. Results by Cohen (1995a, 1995b), Dzeroski et al. (1992) and others indicate that classes of Horn programs having a single or a constant number of clauses are learnable from examples. Khardon shows that “actions strategies” consisting of a variable number of constant-size first-order production rules can be learned from examples (Khardon, 1996). However, Cohen (1995a) proves that even predicting very restricted classes of Horn programs (viz. function-free 0-depth determinate constant arity) with variable number of clauses of variable size from examples alone is cryptographically hard.

Frazier and Pitt (1993) first used the entailment setting for learning arbitrary propositional Horn programs. In addition to examples, they also used entailment membership queries (“entailment queries” from now on) which ask if a Horn clause is entailed by the target. Moving to first order representations, Frazier and Pitt (1993) showed that CLASSIC sentences are exactly learnable in polynomial time from examples and entailment queries. A Horn clause is simple if the terms and the variables in the body of the clause are restricted to the terms that appear in the head. Page (1993) considered non-recursive Horn programs restricted to simple clauses and predicates of constant

*This paper also appears in the proceedings of 8th International Conference on Inductive Logic Programming, 1998 (ILP-98).

arity, and showed that they are learnable from examples and entailment queries. Arimura (1997) generalized Page's result to acyclic (possibly, recursive) simple Horn programs with constant-arity predicates. Reddy and Tadepalli (1997b) showed that function-free non-recursive Horn definitions are learnable from examples and entailment queries. The result we present here applies to non-generative Horn programs, where the variables and the terms in the head are restricted to those in the body. We show that acyclic non-generative Horn programs with constant arity that have polynomial-time subsumption procedure are learnable from examples and entailment queries when certain closure conditions are satisfied. In particular, the result applies to acyclic Horn programs with constant arity determinate clauses.

Goal-decomposition rules are hierarchical in nature, as are Horn programs. One aspect of learning in hierarchical domains is the hierarchical order of literals (goals or concepts). In many systems, learning hierarchically organized knowledge assumes that the structure of hierarchy or the order of the literals is known to the learner. Examples of such work include Marvin (Sammut & Banerji, 1986) and XLearn (Reddy & Tadepalli, 1997a), on the experimental side; learning from exercises by Natarajan (1989) and learning acyclic Horn sentences by Arimura (1997), on the theoretical side. In fact, Khardon shows that learning hierarchical strategies can be computationally hard when the structure of the hierarchy is not known (Khardon, 1996). Our algorithm also assumes that the hierarchical order of the literals is known.

The rest of the paper is organized as follows. Section 2 provides definitions for some of the terminology we use. Section 3 describes the learning model and the learning algorithm, and proves the learnability result. Section 4 concludes the paper with some discussion on implications and limitations of the work.

2 Preliminaries

In this section, we define and describe some of the terminology we use in the rest of the paper. For brevity, we omit some of the standard terminology (as given in books such as (Lloyd, 1987)). In the following, we use p and its variants, and a and its variants each to stand for a conjunction of literals; and b, q, l and their variants each to stand for a single literal.

Definition 1 A definite Horn clause (Horn clause or clause, for short) is a finite set of literals that contains exactly one positive literal— $\{l, \neg l_1, \neg l_2, \dots, \neg l_n\}$. It is treated as a disjunction of the literals in the set with universal quantification over all the variables. Alternately, it is represented as $l_1, l_2, \dots, l_n \rightarrow l$, where

l is called the head or consequent, and l_1, l_2, \dots, l_n is called the body or antecedent and is interpreted as $l_1 \wedge l_2 \wedge \dots \wedge l_n$. A unit Horn clause is a Horn clause with no negative literals and hence no body. A Horn program or Horn sentence is a set of definite Horn clauses interpreted conjunctively.

Definition 2 Let C_1 and C_2 be sets of literals. We say that C_1 subsumes C_2 (denoted $C_1 \succeq C_2$) iff there exists a substitution θ such that $C_1\theta \subseteq C_2$. We also say C_1 is a generalization of C_2 .

Definition 3 (Plotkin, 1970) Let C, C', C_1 and C_2 be sets of literals. We say that C is the least general generalization (lgg) of C_1 and C_2 iff $C \succeq C_1$ and $C \succeq C_2$, and $C' \succeq C$, for any C' such that $C' \succeq C_1$ and $C' \succeq C_2$.

Definition 4 (Plotkin, 1970) A selection of clauses C_1 and C_2 is a pair of literals (l_1, l_2) such that $l_1 \in C_1$ and $l_2 \in C_2$, and l_1 and l_2 have the same predicate symbol, arity, and sign.

If C_1 and C_2 are sets of literals, then $\text{lgg}(C_1, C_2)$ is $\{\text{lgg}(l_1, l_2) : (l_1, l_2) \text{ is a selection of } C_1 \text{ and } C_2\}$. If l is a predicate, $\text{lgg}(l(s_1, s_2, \dots, s_n), l(t_1, t_2, \dots, t_n))$ is $l(\text{lgg}(s_1, t_1), \dots, \text{lgg}(s_n, t_n))$. The lgg of two terms $f(s_1, \dots, s_n)$ and $g(t_1, \dots, t_m)$, if $f = g$ and $n = m$, is $f(\text{lgg}(s_1, t_1), \dots, \text{lgg}(s_n, t_n))$; else, it is a variable x , where x stands for the lgg of that pair of terms throughout the computation of the lgg of the set of literals.

As an example, let C_1 be $l(a, b), l(b, c), m(b) \rightarrow l(a, c)$, and C_2 be $l(1, 2), l(2, 3), m(2) \rightarrow l(1, 3)$. $(l(a, c), l(1, 3))$ and $(\neg m(b), \neg m(2))$ are two of the selections of C_1 and C_2 . $\text{lgg}(C_1, C_2)$ is $l(x, y), l(y, z), l(t, u), l(v, w), m(y) \rightarrow l(x, z)$, where x, y, z, t, u, v and w are variables standing for the pairs $(a, 1), (b, 2), (c, 3), (a, 2), (b, 3), (b, 1)$ and $(c, 2)$.

Definition 5 A derivation of a Horn clause $p \rightarrow q$ from a Horn program H is a finite directed acyclic graph G such that there is a node q , there is no arc (q, r) in G , and for each node l in G , either $l \in p$ or if $(l_1, l), \dots, (l_d, l)$ are the only arcs of G terminating at l , then $l_1, \dots, l_d \rightarrow l = C\theta$ for some clause $C \in H$ and a substitution θ .

For example, let H be $\{\text{parent}(x, y), \text{parent}(y, z) \rightarrow \text{grandParent}(x, z); \text{mother}(x, y) \rightarrow \text{parent}(x, y)\}$. Figure 1 shows a derivation of $\text{mother}(a, b), \text{mother}(b, c) \rightarrow \text{grandParent}(a, c)$.

Proposition 1 In a derivation G of a clause $p \rightarrow q$ from a Horn program H , for any node l , either l is in p or $H \models p \rightarrow l$.

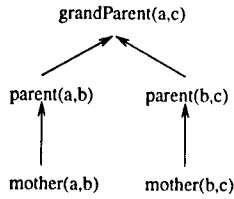


Figure 1: A derivation of $mother(a,b)$, $mother(b,c) \rightarrow grandParent(a,c)$ from H .

Let P be a set of predicate symbols, and T be a set of terms. Let L be a set of atoms defined using P and T . Let \mathcal{H} be a set of Horn programs using atoms in L only. If k is an integer, then P_k is a subset of P containing only those predicate symbols of arity k or less. Further, L_k is a set of atoms defined using P_k and T , and \mathcal{H}_k is a set of Horn programs using atoms in L_k only. In the following three definitions, we describe a class of Horn programs AH_k for which minimal models are of polynomial size.

Definition 6 (Arimura, 1997) Let $\Sigma \in \mathcal{H}$. Then a binary relation supported by (denoted, \succ) over atoms in L w.r.t. Σ is such that (1) for all $p \rightarrow l \in \Sigma$, and for all $l_1 \in p$, $l \succ l_1$; (2) for all $l_1, l_2 \in L$ and every substitution θ , if $l_1 \succ l_2$, then $l_1\theta \succ l_2\theta$; and (3) if $l_1 \succ l_2$ and $l_2 \succ l_3$ then $l_1 \succ l_3$.

Definition 7 A Horn program Σ is acyclic over L if the relation \succ over L w.r.t. Σ is terminating; i.e., for any $l \in L$, there is no infinite decreasing sequence $l \succ l_1 \succ \dots$.

In the last example, H is acyclic because $grandParent(x,y) \succ parent(x,y) \succ mother(x,y)$ and there is no cycle formed by the \succ relation.

Following Khardon (1998), we call a definite clause a *non-generative clause* if the set of terms in its consequent are a subset of the set of terms and subterms in its antecedent.

Definition 8 If k is a constant, we define a Horn program $\Sigma \in \mathcal{H}_k$ to be in the class AH_k , if Σ is acyclic over L_k , and each clause is either non-generative or has an empty antecedent.

Definition 9 Let $a \rightarrow b$ be a clause in a Horn program Σ , and $p \rightarrow q$ be a clause. Then, $a \rightarrow b$ is a **target clause** in Σ of $p \rightarrow q$ iff $a \rightarrow b \succeq p \rightarrow q$, i.e., for a substitution θ , $a\theta \subseteq p$, $b\theta = q$. We call $p \rightarrow q$ a **hypothesis clause** of $a \rightarrow b$.

Definition 10 For an antecedent p , q' is a **prime consequent** of p wrt Σ if $\Sigma \models p \rightarrow q'$, $q' \not\subseteq p$, and there is no $l \in L$ such that $q' \succ l$, $\Sigma \models p \rightarrow l$ and $l \not\subseteq p$.

In the last example, $parent(a,b)$ is a prime consequent of $mother(a,b)$, $mother(b,c)$, but $grandParent(a,c)$ is not—since $parent(a,b) \succ grandParent(a,c)$.

3 Learning Horn Programs

In this section, we show that a subclass of AH_k is exactly learnable, using the exact learning model (Angluin, 1988), in entailment setting. Henceforth, $\Sigma \in AH_k$ denotes a target Horn program.

3.1 The Learning Model

In learning from entailment, an example is a Horn clause. An example $p \rightarrow q$ is a positive example of Σ if $\Sigma \models p \rightarrow q$; negative, otherwise. An *entailment query* takes as input an example $(p \rightarrow q)$, and outputs *yes* if it is a positive example of Σ ($\Sigma \models p \rightarrow q$), and *no* otherwise. An *equivalence query* takes as input a Horn program H and outputs *yes* if H and Σ contain (entail) exactly the same Horn clauses; otherwise, returns a *counterexample* that is in (entailed by) exactly one of H and Σ . A *derivation-order query*, \succ , takes as input two atoms l_1 and l_2 in L and outputs *yes* if $l_1 \succ l_2$, and *no* otherwise. An algorithm *exactly learns* a Horn program Σ in AH_k in polynomial time from equivalence, entailment, and derivation-order (\succ) queries if and only if it runs in time polynomial in the size of Σ and in the size of the largest counterexample, and outputs a Horn program in AH_k such that equivalence query answers *yes*.

3.2 The Learning Algorithm

In this section, we describe the learning algorithm, **PLearn**, shown in Figure 2. **PLearn** always maintains a hypothesis H which is entailed by the target, so that every instance of H is also an instance of Σ and all counterexamples are positive.

Suppose that a counterexample $p \rightarrow q$ is given to the learner—see Figure 2. Every such counterexample has a derivation from the target theory, Σ . Since this derivation is not possible from the current hypothesis H , there is some clause used in the derivation that has not been learned with sufficient generality. The algorithm tries to identify the antecedent literals of such a clause, c^* , in the target by expanding the derivation graph from its leaves in p toward the goal using the clauses in H . In other words, **PLearn** computes the minimal model (p'_f) of H implied by p (“closure” or “saturation”) by forward chaining (line 4). To identify the consequent of c^* , also called the “prime consequent” of p'_f , **PLearn** calls **PrimeCons** in line 5. **PrimeCons** finds the prime consequent of p'_f by tracing the “supported-by” chain starting from q for a literal q_f not in p'_f , but is directly supported by some of the lit-

PLearn

Given equivalence, entailment and \succ queries
outputs a Horn program H s.t. **equivalent?**(H, Σ) is Yes.

```
(1)  $H = \{\}$  /* empty hypothesis-clauses set */
(2) while not equivalent?( $H, \Sigma$ ) do {
(3)   Let  $p \rightarrow q$  be the counterexample returned
(4)    $p'_f = \{l : H \models (p \rightarrow l)\}$  /* forward chaining */
(5)    $q_f = \text{PrimeCons}(p'_f \rightarrow q)$ 
(6)    $p_f \rightarrow q_f = \text{Reduce}(p'_f \rightarrow q_f)$ 
(7)   if  $\exists p_i \rightarrow q_i \in H$  such that  $\Sigma \models p_g \rightarrow q_g$ ,
(8)     where  $p_g \rightarrow q_g = \text{lgg}(p_i \rightarrow q_i, p_f \rightarrow q_f)$ 
(9)   then replace first such  $p_i \rightarrow q_i$  by  $\text{Reduce}(p_g \rightarrow q_g)$ 
(10)  else append  $p_f \rightarrow q_f$  to  $H$ 
(11) } /* while */
(12) return  $H$ 
```

PrimeCons($p \rightarrow q$) /* finds prime consequents */

```
(13) Let  $L$  be the set of all possible literals having
      only those terms that are in  $p$ 
(14)  $q' = q$ ;
(15)  $L' = \{l : l \in L - p \text{ and } \Sigma \models p \rightarrow l\}$ 
(16) while  $\exists l \in L'$  such that  $q' \succ l$ 
(17)    $q' = l$ ;
(18) return  $q'$ 
```

Reduce($p \rightarrow q$) /* trims irrelevant literals */

```
(19)  $p' = p$ 
(20) repeat
(21)   for each literal  $l$  in  $p'$  in sequence do
(22)     if  $\Sigma \not\models (p' - \{l\}) \rightarrow l$  and  $\Sigma \models (p' - \{l\}) \rightarrow q$ 
(23)     then  $p' = p' - \{l\}$ 
(24) until there is no change to  $p'$ 
(25) return  $p' \rightarrow q$ 
```

Figure 2: PLearn Algorithm

erals in p'_f (lines 13–18). In line 6, PLearn makes use of Reduce to trim away “irrelevant” literals from the antecedent p'_f to form a new clause $p_f \rightarrow q_f$ that is also a counterexample to the hypothesis and is subsumed by a single target clause—see Lemmas 9,2,3. PLearn combines $p_f \rightarrow q_f$ with an “appropriate” clause $p_i \rightarrow q_i$ in H using *lgg* (lines 7–9). It uses the entailment query to find an appropriate hypothesis clause by checking if the result of *lgg* is implied by the target (line 7). If no such clause exists in H , $p_f \rightarrow q_f$ is appended to H as a new clause (line 10).

One problem with this approach is that the size of the *lgg* is a product of the sizes of its two arguments. This causes the size of a hypothesis clause to grow exponentially in the number of examples combined with it in the worst case. To avoid this, the antecedent literals of the clause after *lgg* are again trimmed using Reduce so that the size of the resulting clause is bounded, while it is still subsumed by the target clause (lines 19–25). The result of Reduce then replaces the original hypothesis clause $p_i \rightarrow q_i$ it is derived from (line 9). After this step, only the antecedents of the target clause and some of their consequents remain in

the resulting hypothesis clause—see Lemma 5. This process repeats until the hypothesis H is equivalent to Σ . The algorithm works for unit clauses (which have empty antecedents) without change.

3.3 An Example

As an example to see how PLearn works, consider $\Sigma = \{l_1(f(x)), l_2(x), l_3(x) \rightarrow l_4(x); l_1(f(x)), l_2(x) \rightarrow l_5(x); l_4(x), l_5(x) \rightarrow l_7(x)\}$ where f is a function symbol. Suppose $H = \{l_1(f(c)), l_2(c) \rightarrow l_5(c)\}$. We adopt the convention that the letters such as a, b, c , etc. at the beginning of the alphabet are constants and the letters at the end of the alphabet such as x, y, z , etc. are variables. Let the counterexample be $l_1(f(d)), l_2(d), l_3(d) \rightarrow l_7(d)$. In step 4, it does not change. In PrimeCons, since $l_7(d) \succ l_4(d)$ and $l_7(d) \succ l_5(d)$, $l_7(d)$ is not a prime consequent, but any one of $l_4(d)$ and $l_5(d)$ is. Suppose PrimeCons returns $l_5(d)$. Reduce eliminates $l_3(d)$ from the antecedent, because $\Sigma \models l_1(f(d)), l_2(d) \rightarrow l_5(d)$, and $\Sigma \not\models l_1(f(d)), l_2(d) \rightarrow l_3(d)$. Thus, $p_f \rightarrow q_f = l_1(f(d)), l_2(d) \rightarrow l_5(d)$. Combining this with the clause in H , we obtain $p_g \rightarrow q_g = l_1(f(x)), l_2(x) \rightarrow l_5(x)$ is entailed by Σ , new H is $\{l_1(f(x)), l_2(x) \rightarrow l_5(x)\}$.

Suppose the next counterexample is $l_1(f(c)), l_2(c), l_3(c) \rightarrow l_7(c)$. Then, $q_f = l_4(c)$, and $p'_f = \{l_1(f(c)), l_2(c), l_3(c), l_5(c)\}$. $p_f \rightarrow q_f = l_1(f(c)), l_2(c), l_3(c), l_5(c) \rightarrow l_4(c)$, since Reduce cannot remove $l_5(c)$, because it is implied by the other literals wrt Σ (line 22). The modified counterexample $p_f \rightarrow q_f$ cannot be combined with the clause in H , because the resultant $p_g \rightarrow q_g$ after *lgg*, $l_1(f(x)), l_2(x) \rightarrow$, is not entailed by Σ . Hence, it is appended to H to make $H = \{l_1(f(x)), l_2(x) \rightarrow l_5(x); l_1(f(c)), l_2(c), l_3(c), l_5(c) \rightarrow l_4(c)\}$.

Suppose the next counterexample is again $l_1(f(c)), l_2(c), l_3(c) \rightarrow l_7(c)$. After line 4, $p'_f = l_1(f(c)), l_2(c), l_3(c), l_5(c), l_4(c)$. q_f now is $l_7(c)$, because it is a prime consequent of p'_f . After Reduce, $p_f = l_5(c), l_4(c)$. $p_f \rightarrow q_f$ cannot be combined with the clauses in H , because the resultant *lgg*'s are not entailed by Σ . Again, $p_f \rightarrow q_f$ is added to H . This process continues until H and Σ are equivalent.

To bring out the nuances in Reduce, let us revisit the last part of the previous example. Consider the input $l_5(c), l_2(c), l_3(c), l_1(f(c)), l_4(c) \rightarrow l_7(c)$ to Reduce. Although $\Sigma \models l_1(f(c)), l_2(c), l_3(c) \rightarrow l_7(c)$, since $\Sigma \models l_1(f(c)), l_2(c), l_3(c) \rightarrow l_4(c)$ and $\Sigma \models l_1(f(c)), l_2(c) \rightarrow l_5(c)$, the literal $l_5(c)$ cannot be removed. This is because $l_5(c)$ is implied by the other literals ($l_1(f(c)), l_2(c)$) wrt Σ . The order in which the literals are removed in Reduce follows the derivation order: if $l_i \succ l_j$, if at all l_i is removed, it is removed after

l_j is removed. This can be intuitively imagined in the following way. Consider a derivation tree for a counterexample, with the consequent literal on top and the antecedent literals at the bottom. The above process trims off the literals bottom-up in the tree up to the appropriate level, so that the resulting clause is subsumed by some clause in the target. In the above case, if Reduce removes $l_5(c)$ and leaves over $l_1(f(c)), l_2(c)$, the resulting clause $(l_1(f(c)), l_2(c), l_3(c), l_4(c) \rightarrow l_7(c))$ is not subsumed by any clause in Σ .

However, this means that Reduce leaves over literals which are implied by the remaining literals, i.e., l cannot be removed from p' if $\Sigma \models (p' - \{l\}) \rightarrow l$ (line 22). Removing such literals could result in hypothesis clauses which are not subsumed by any target clause, as the following example illustrates. Let Σ be $\{l_1(a) \rightarrow l_2(a); l_1(x), l_2(x) \rightarrow l_3(x)\}$. Suppose the first counterexample is $l_1(a), l_2(a) \rightarrow l_3(a)$. Hence $p'_f = \{l_1(a), l_2(a)\}$ and $q_f = l_3(a)$ in line 6. If Reduce were to remove $l_2(a)$ from p'_f because $\Sigma \models l_1(a) \rightarrow l_3(a)$, it ends up with a clause that is not subsumed by any target clause. We would like to prevent such redundant hypothesis clauses so that their number is not too high compared to the number of target clauses. (This argument is formalized in Lemmas 6, 7 and 8.)

3.4 Learnability of AH_k

In this section, we prove that PLearn algorithm in Figure 2 exactly learns a subclass of AH_k for which subsumption is of polynomial-time complexity. The plan of the proof is as follows: Through a series of lemmas, we first establish that every hypothesis clause learned has a target clause (Lemma 6). We then show that every target clause has at most one hypothesis clause (Lemma 8). Together, these two lemmas establish that the number of hypothesis clauses is bounded by the number of target clauses. We use this fact and the bounds of the sizes on the hypothesis clauses (established in Lemma 5) to show that PLearn learns successfully in polynomial time (Theorems 10 and 11). We then define a specific hypothesis class that obeys the conditions of these theorems and prove that this class is learnable (Theorem 12).

Lemmas 2 and 3 show that PrimeCons with the input $p \rightarrow q$ finds a (prime) consequent q' of p such that $p \rightarrow q'$ is subsumed by a clause in Σ .

Lemma 2 *Let $p \rightarrow q$ be the input and q' be the output of PrimeCons. Assume that $q \notin p$ and $\Sigma \models p \rightarrow q$. Then, (1) PrimeCons terminates; (2) q' is a prime consequent of p wrt Σ .*

Proof. (1) Since Σ is acyclic, there is a terminating sequence $q \succ l_1 \succ l_2 \dots$. Since the loop of lines 16–17 can only iterate as many times as the length of the

sequence, PrimeCons terminates.

(2) q' is such that $\Sigma \models p \rightarrow q'$, and $q' \notin p$ (by lines 15–17). Since q' is as in line 17 in the iteration immediately prior to the terminating iteration of lines 16–17, there is no l such that $q' \succ l$, $\Sigma \models p \rightarrow l$ and $l \notin p$. Thus, q' is a prime consequent of p wrt Σ . \square

Lemma 3 *If q' is a prime consequent of p wrt Σ , then there is a clause $C \in \Sigma$ such that $C \succeq p \rightarrow q'$.*

Proof. Assume that q' is a prime consequent of p wrt Σ . Consider a derivation G of $p \rightarrow q'$ in Σ . Let $(l_1, q'), \dots, (l_d, q')$ be the only arcs of G that terminate at q' . This implies that $q' \succ l_i$ for all $l_i \in \{l_1, \dots, l_d\}$. It must be that every l_i is in p ; otherwise, there is an l (viz. l_i) such that $q' \succ l$, $\Sigma \models p \rightarrow l$ and $l \notin p$ —contradicting the assumption that q' is a prime consequent. Thus, $\{l_1, \dots, l_d\} \subseteq p$. But, $l_1, \dots, l_d \rightarrow q' = C\theta$ for some clause $C \in H$ and a substitution θ , following the definition of derivation. Thus, $C\theta \subseteq p \rightarrow q'$, implying that $C \succeq p \rightarrow q'$. \square

The following definition and Lemmas 4 and 5 help show that Reduce, given a clause $p \rightarrow q$ as input, removes irrelevant literals from antecedent p , while maintaining q as a consequent.

Definition 11 *If a is a conjunction, closure of a with respect to Σ , denoted by κ_a , is defined as $\{l | \Sigma \models (a \rightarrow l)\}$.*

Lemma 4 *If q is a prime consequent of p and $p' \rightarrow q = \text{Reduce}(p \rightarrow q)$, then q is a prime consequent of p' also.*

Proof. Because q is a prime consequent of p and $p' \subseteq p$, any literal other than the ones in $p - p'$, cannot be prime consequents of p' . By lines 22–23, only those literals l that are not supported by p' are removed. In which case, no literal l in $p - p'$, can be such that $\Sigma \models p' \rightarrow l$. Hence, q is a prime consequent of p' as well. \square

Lemma 5 *If the input $p \rightarrow q$ to Reduce is s.t. q is a prime consequent of p wrt Σ , then the output $p' \rightarrow q$ is such that $p' \subseteq \kappa_{a\theta}$ where $a \rightarrow b$ is a clause in Σ and $a\theta \subseteq p'$ and $b\theta = q$.*

Proof. Since q is a prime consequent of p , by Lemma 4, q is a prime consequent of p' also. Then, by Lemma 3, there is a clause $a \rightarrow b \in \Sigma$, and a θ such that $a\theta \subseteq p'$ and $b\theta = q$. We now show that $p' \subseteq \kappa_{a\theta}$. Assume that there exists a literal in $p' - \kappa_{a\theta}$. Let $l \in p' - \kappa_{a\theta}$ be a least such literal so that there is no literal l' in $p' - \kappa_{a\theta}$ such that $l \succ l'$. Such a literal must exist, because Σ is acyclic. There are two reasons for l to remain in $p' - \kappa_{a\theta}$: either (a) $\Sigma \not\models (p' - \{l\}) \rightarrow q$

or (b) $\Sigma \models (p' - \{l\}) \rightarrow l$. We disprove both the cases:
 (a) Since $a\theta \subseteq p'$, and l is not in $\kappa_{a\theta}$ and thus not in $a\theta$, $a\theta \subseteq (p' - \{l\})$. Therefore, $\Sigma \models (p' - \{l\}) \rightarrow q$.
 (b) The only other reason why l remains in p' is that $\Sigma \models (p' - \{l\}) \rightarrow l$. That means that $p' - \{l\}$ contains literals that imply l . There must be at least one such literal in p' that is not in $\kappa_{a\theta}$, or else $l \in \kappa_{a\theta}$, contradicting $l \in p' - \kappa_{a\theta}$. But then $p' - \kappa_{a\theta}$ contains literals l' such that $l \succ l'$, which contradicts the statement that there is no such l' . Thus, we disprove both the possibilities. Hence, $p' \subseteq \kappa_{a\theta}$. \square

Lemmas 6, 7 and 8, below, show that PLearn only maintains right clauses in H .

Lemma 6 Every clause $p_i \rightarrow q_i \in H$ has a target clause.

Proof. We first show that each $p_i \rightarrow q_i \in H$ is such that q_i is a prime consequent of p_i . Then, by Lemma 3, $p_i \rightarrow q_i$ has a clause $C \in \Sigma$ such that $C \succeq p_i \rightarrow q_i$.

We show that q_i is a prime consequent of p_i by induction on the number of times a clause at position i in H is updated. It is first introduced by line 10. By Lemmas 2 and 4, q_f is a prime consequent of p_f . This proves the base case. The other way a clause becomes a hypothesis clause is by line 9. The clause at position i in H ($p_i \rightarrow q_i$) is updated by line 9. As inductive hypothesis, assume that each $p_i \rightarrow q_i$ in H is such that q_i is a prime consequent of p_i , at the beginning of an iteration of the loop of lines 2–11 when position i in H is updated. Consider $p_g \rightarrow q_g = \text{lgg}(p_i \rightarrow q_i, p_f \rightarrow q_f)$. Suppose q_g is not a prime consequent of p_g , but q'_g such that $q_g \succ q'_g$ is. Let θ_f and θ_i be substitutions such that $p_g\theta_f \subseteq p_f$, $q_g\theta_f = q_f$, $p_g\theta_i \subseteq p_i$, and $q_g\theta_i = q_i$. Let $q'_f = q'_g\theta_f$ and $q'_i = q'_g\theta_i$. Since $q_g \succ q'_g$, by the definition of \succ order, $q_f \succ q'_f$ and $q_i \succ q'_i$. Since q_f is a prime consequent of p_f , q'_f must be in p_f . Similarly, q'_i must be in q_i . Therefore, $\text{lgg}(q'_i, q'_f) = q'_g$ must be in p_g , contradicting the assumption that q'_g is a prime consequent of p_g . Hence, q_g is a prime consequent of p_g . By Lemma 4 if $p_i \rightarrow q_i = \text{Reduce}(p_g \rightarrow q_g)$, then q_i is a prime consequent of p_i . So by Lemma 3, $p_i \rightarrow q_i$ has a target clause. \square

Lemma 7 If PLearn combines a modified counterexample $p_f \rightarrow q_f$ with a clause $p_i \rightarrow q_i \in H$, then there is a target clause C s.t. $C \succeq p_f \rightarrow q_f$ and $C \succeq p_i \rightarrow q_i$. Further, there is no C' s.t. $C' \succeq p_j \rightarrow q_j$ and $C' \succeq p_f \rightarrow q_f$, for any $j < i$.

Proof. PLearn combines $p_f \rightarrow q_f$ with $p_i \rightarrow q_i$ only if $\Sigma \models \text{lgg}(p_i \rightarrow q_i, p_f \rightarrow q_f)$. By Lemma 6, q_g is a prime consequent of p_g where $p_g \rightarrow q_g = \text{lgg}(p_i \rightarrow q_i, p_f \rightarrow q_f)$. By Lemma 3, there is a $C \in \Sigma$ such that $C \succeq p_g \rightarrow q_g$. Hence, $C \succeq p_i \rightarrow q_i$ and $C \succeq p_f \rightarrow q_f$.

Since $p_f \rightarrow q_f$ is combined with $p_i \rightarrow q_i$, for any $j < i$, $\Sigma \not\models \text{lgg}(p_j \rightarrow q_j, p_f \rightarrow q_f)$. Therefore, there is no C' s.t. $C' \succeq \text{lgg}(p_j \rightarrow q_j, p_f \rightarrow q_f)$. Thus, there is no C' s.t. $C' \succeq p_j \rightarrow q_j$ and $C' \succeq p_f \rightarrow q_f$. \square

Lemma 8 Every clause $C \in \Sigma$ has at most one hypothesis clause.

Proof. First, we show that any new hypothesis clause added to H has a target clause distinct from the target clauses of the other hypothesis clauses in H . Next, we show that if two hypothesis clauses do not have common target clauses at the beginning of an iteration of the loop of lines 2–11, then they still have distinct target clauses at the end of the iteration.

When $p_f \rightarrow q_f$ is added to H , by Lemma 7, for any clause H_i in H , there is no $C \in \Sigma$ such that $C \succeq H_i$ and $C \succeq p_f \rightarrow q_f$. Therefore, $p_f \rightarrow q_f$, a new clause added to H , has a target clause distinct from the target clauses of the other hypothesis clauses then in H . Next, at most one of H_i and H_j can change in an iteration of the loop. If neither changes, we are done with the proof. Suppose that H_i changes, without loss of generality. Let C be any target clause of H_j . Assume that H_i and H_j do not have a common target clause at the beginning of an iteration. Hence, C is not a target clause of H_i . That is, $C \not\preceq H_i$. Let e be the counterexample for the current iteration. We first show that $\text{lgg}(H_i, e)$ does not have C as a target clause. Since $C \not\preceq H_i$, $C \not\preceq \text{lgg}(H_i, e)$. Therefore, C is not a target clause of $\text{lgg}(H_i, e)$. Let $\text{lgg}(H_i, e)$ be $p_g \rightarrow q_g$, and C be $a \rightarrow b$. Hence, for every θ , either $a\theta \not\subseteq p_g$ or $b\theta \neq q_g$. If $a\theta \not\subseteq p_g$, $a\theta$ is not a subset of any subset of p_g . Since Reduce outputs a clause with a subset of p_g as the antecedent and q_g as the consequent, $C \not\preceq \text{Reduce}(\text{lgg}(H_i, e))$. Therefore, H_j and the new clause in position i , $\text{Reduce}(\text{lgg}(H_i, e))$, do not have a common target clause even at the end of the iteration. \square

The following lemma shows that even after the modifications due to PrimeCons and Reduce counterexample remains a counterexample.

Lemma 9 $p_f \rightarrow q_f$ as in line 6 of PLearn is a positive counterexample.

Proof. First, we show that every counterexample $p \rightarrow q$, as in line 3, is a positive counterexample. Then, we argue that $p'_f \rightarrow q_f$ (lines 4 and 5) is also a positive counterexample. Finally, we show that $p_f \rightarrow q_f$ (line 6) is a positive counterexample.

Since, by Lemma 6, for every $H_i \in H$, there is a clause $C \in \Sigma$ such that $C \succeq H_i$, $\Sigma \models H$. Therefore, $p \rightarrow q$, as in line 3, is a positive counterexample. Since $p \subseteq p'_f$, $\Sigma \models p'_f \rightarrow q$. Since p'_f contains all and only those

literals l such that $H \models p \rightarrow l$, for any literal $l' \notin p'_f$, $H \not\models p'_f \rightarrow l'$. Since q_f (by lines 5 and 15) is not in p'_f , $H \not\models p'_f \rightarrow q_f$. By line 15, $\Sigma \models p'_f \rightarrow q_f$. Therefore, $p'_f \rightarrow q_f$ is also a positive counterexample. Finally, since $p_f \subseteq p'_f$, $H \not\models p_f \rightarrow q_f$. By lines 6 and 22, $\Sigma \models p_f \rightarrow q_f$. Thus, $p_f \rightarrow q_f$ is a positive counterexample. \square

Finally, Theorem 10 shows that PLearn exactly learns AH_k when forward chaining using H is of polynomial-time complexity. Theorem 11 identifies conditions on Σ such that PLearn returns an H for which time complexity of forward chaining is polynomial.

Theorem 10 *PLearn exactly learns AH_k with equivalence, \succ , and entailment queries, provided determining $H \models p \rightarrow l$ is polynomial in the sizes of H and p .*

Proof. By Lemma 9, $p_f \rightarrow q_f$ is a positive counterexample. For each counterexample, either a new antecedent is added (line 10) or an existing antecedent is replaced (line 9). In the latter case, the replaced clause $p_i \rightarrow q_i$ must be subsumed by the replacing clause $p' \rightarrow q_g$, since both lgg and Reduce generalize the original clause by turning constants to variables and dropping literals. On the other hand, the replaced clause must not subsume (and hence be different from) the replacing clause $p' \rightarrow q_g = \text{Reduce}(p_g \rightarrow q_g)$. If not, that is if $p_i \rightarrow q_i \succeq p' \rightarrow q_g$, since $p' \rightarrow q_g \succeq p_g \rightarrow q_g \succeq p_f \rightarrow q_f$, $p_i \rightarrow q_i \succeq p_f \rightarrow q_f$. Since $p_i \rightarrow q_i \in H$, $H \models p_f \rightarrow q_f$ —thus contradicting that $p_f \rightarrow q_f$ was a counterexample of H . Hence, the replacement at a position in H changes the clause at that position. The minimum change there can be is either a variablization of a constant or a removal of a literal.

Let n be the number of clauses, and s be the number of distinct predicate symbols in Σ . Further, let the maximum number of terms in any clause be t , and in any counterexample be t_e .

The maximum possible number of literals there can be using t terms is at most st^k . Hence, the maximum number of literals in κ_a , and therefore, by Lemmas 5 and 6, in each clause is at most st^k . This includes all literals and their variablized versions. Hence, we can consider variablization as removing a literal. Thus, we need at most st^k counterexamples for each clause. (This includes one base counterexample to introduce a clause into H .) By Lemmas 6 and 8, there are at most n clauses in H . Hence, we need at most nst^k counterexamples or equivalence queries. A call to PrimeCons from line 5 takes at most st_e^k entailment queries, because the literals we need to try as possible consequents are all in L , and $|L| \leq st_e^k$. PrimeCons is called once for each of the counterexamples.

For each of the nst^k counterexamples, the condition

in line 7 is tested at most n times, which needs at most n entailment queries. Reduce is called with the argument $p'_f \rightarrow q_f$ once for each of the counterexamples, and with the arguments $p_g \rightarrow q_g$ for at most nst^k counterexamples. In $\text{Reduce}(p \rightarrow q)$, in $|p|$ iterations of the loop of lines 21–23, at least one literal is removed. So, this loop can be tried at most $|p|$ times. Each iteration of the loop of lines 21–23 takes two entailment queries. Therefore, $\text{Reduce}(p \rightarrow q)$ needs at most $|p|(|p| + 1)$ entailment queries. Hence, $\text{Reduce}(p'_f \rightarrow q_f)$ needs at most $n_f = st_e^k(st_e^k + 1)$ entailment queries. Since $p_i \rightarrow q_i$ and $p_f \rightarrow q_f$ are outputs of Reduce, the maximum possible number of literals in $p_g \rightarrow q_g = lgg(p_i \rightarrow q_i, p_f \rightarrow q_f)$ is at most s^2t^{2k} . Hence, $\text{Reduce}(p_g \rightarrow q_g)$ needs at most $n_g = s^2t^{2k}(s^2t^{2k} + 1)$ entailment queries. Thus, the total number of entailment queries is at most $nst^k(st_e^k + n + n_f + n_g)$.

If determining $H \models (p \rightarrow l)$ takes $\mathcal{P}(n, l, t_e)$ time where \mathcal{P} is a polynomial, then line 4 takes at most $st_e^k \cdot \mathcal{P}(n, l, t_e)$ time. In the rest, the number of entailment queries dominates the time. Hence, the time taken by PLearn is polynomial in n, s, l, v, t , and t_e . \square

Definition 12 *Let $p \rightarrow q$ be a Horn clause. $p' \rightarrow q$ is called its antecedent expansion if $p \subseteq p'$ and p' contains only those variables in p . A class C of Horn sentences is closed under antecedent expansion, if every Horn sentence obtained by selecting a subset of its Horn clauses and replacing them with their antecedent expansions is also in C .*

Definition 13 *A subsumption algorithm takes a clause $a \rightarrow b$, a conjunction of literals p , and a ground substitution θ for the variables in b , and returns true if and only if $a\theta \succeq p$.*

Theorem 11 *PLearn exactly learns a subclass C of AH_k with equivalence, \succ , and entailment queries, provided that (a) C is closed under substitution and antecedent expansion and (b) the clauses $a \rightarrow b$ of the target concepts in C have a polynomial-time subsumption algorithm.*

Proof. By Lemma 5, each clause $p_i \rightarrow q_i \in H$ in PLearn has a target clause $a \rightarrow b$ and a substitution θ such that $a\theta \subseteq p_i \subseteq \kappa_{a\theta}$. Since the target class is closed under substitution and antecedent expansion, the hypothesis clauses have a polynomial-time subsumption algorithm. Hence, the forward-chaining step of computing the consequents of p in line 4 of PLearn can be done in polynomial time by repeatedly checking for a hypothesis clause $a \rightarrow b$ whose antecedent subsumes p after a substitution θ of the variables in b , and adding $b\theta$ to p . Hence, by the previous theorem, PLearn exactly learns C . \square

The following definition and theorem identify some syntactic restrictions on AH_k such that the resulting subclass satisfies the conditions of the previous theorem.

Definition 14 Let p be a set of literals. A Horn clause $l_1, \dots, l_n \rightarrow q$ is *i-determinate w.r.t. p* iff there exists an ordering l_{o_1}, \dots, l_{o_n} of l_1, \dots, l_n such that for every $i < j \leq n$ and every substitution θ such that $(l_{o_1}, \dots, l_{o_{j-1}} \rightarrow q)\theta$ is ground and $\{l_{o_1}, \dots, l_{o_{j-1}}\}\theta \subseteq p$, there is at most one substitution α for the variables in $l_{o_j}\theta$ such that $l_{o_j}\theta\alpha$ is ground and is in p .¹ We call such an ordering of the literals in the clause an *i-determinate ordering w.r.t. p* . A Horn program is *i-determinate w.r.t. p* iff each of the clauses in the program is *i-determinate w.r.t. p* .

Theorem 12 The class of *i-determinate Horn programs* in AH_k , denoted as $iDetAH_k$, is exactly learnable with equivalence, \succ , and entailment queries.

Proof. First we show that $iDetAH_k$ is closed under substitution and antecedent expansion. Consider a target clause $(l_1, \dots, l_n \rightarrow q)$ for a target program in $iDetAH_k$, whose antecedent literals are sorted in the determinate order. Let $(l_1, \dots, l_n, l_{n+1}, \dots, l_m \rightarrow q)\beta$ be the target clause after antecedent expansion and substitution. We want to show the new clause to be *i-determinate*.

For every set of literals p , substitution θ , and j such that $i < j \leq m$ and $(l_1, \dots, l_{j-1})\beta\theta \subseteq p$ is ground, there is a substitution γ which is equivalent to applying β and θ one after another so that $(l_1, \dots, l_{j-1})\beta\theta = (l_1, \dots, l_{j-1})\gamma$ and $l_j\beta\theta = l_j\gamma$ for any l_j . Since the target clause satisfies *i-determinacy*, there must be at most a single ground substitution α for $l_j\gamma$, $j \leq n$, so that $l_j\gamma\alpha \in p$, which means that this is true for $l_j\beta\theta$ as well. Since the literals from l_{n+1} through l_m do not have any variables not already in l_1 through l_n , there is at most a single ground substitution for them as well. Hence, $(l_1, \dots, l_m \rightarrow q)\beta$ is also *i-determinate*.

Now we show that the clauses of the programs in $iDetAH_k$ have a polynomial-time subsumption algorithm. Given a set of literals p and a clause $l_1, \dots, l_n \rightarrow q$ (whose literals have an unknown determinate ordering), consider all possible subsets of $\{l_1, \dots, l_n\}$ of size i and less. Note that there are at most $O(n^i)$ such subsets. For each such subset, instan-

tiate all the ki variables in that subset in all possible ways. If the total number of terms in p and Σ is t , this gives us t^{ki} different substitutions. For each such substitution, there is at most one substitution for the remaining literals in the clause. The order in which the remaining literals have to be substituted can be determined by sequential search—apply the current substitution to each literal and pick the one that only allows one possible substitution for its remaining variables. This can be done in $O(n^2|p|)$ time. If the antecedent l_1, \dots, l_n subsumes p , then one of the considered subsets should yield a successful match. Hence, the total time for the algorithm is bounded by $O(n^i t^{ki} n^2 |p|)$, which is polynomial in all variables except k and i which are assumed to be constants.

Since the class $iDetAH_k$ satisfies the two conditions required by Theorem 11 for PLearn to be successful, the result follows. \square

4 Discussion and Conclusions

In this paper, we have shown the learnability of certain subclasses of acyclic k -ary Horn programs. More specifically *i-determinate Horn programs* in AH_k , are exactly learnable with equivalence and entailment queries. Unlike the work of Page (1993) and Arimura (1997), the programs we considered allow local variables in the antecedents. However, the clauses must be non-generative in that the set of terms and variables that occur in the head of the clause must be a subset of those that occur in the body of the clause. This is needed to constrain the forward-chaining inference step to finish in polynomial-time, which could otherwise become unbounded. It appears that simultaneously removing both the non-generative and simplicity restrictions could be difficult when functions are present, due to the unbounded nature of inference in that case.

Learning from entailment and *learning from interpretations* are two of the standard settings for first-order learning (De Raedt, 1997). In learning from interpretations, the learner is given a positive (or negative) interpretation for which the Horn sentence is true (or false). Interpretations can be partial in that the truth values of some ground atoms may be left unspecified. When membership queries are available, learning from entailment and learning from interpretations are equivalent for Horn programs. Hence we can use PLearn to learn from (negative) interpretations as follows. Given a negative interpretation, “minimize” it by removing the negative literals from it and asking membership queries. Since every negative interpretation must violate some Horn clause, this yields an interpretation with a set of positive literals l_1, \dots, l_n and at most one negative literal q_i . We

¹This definition strictly generalizes the standard definition of determinacy (Muggleton & Feng, 1990), in that a Horn clause (program) is determinate w.r.t. a set of literals p when it is 0-determinate w.r.t. p . *i-determinacy* should not be confused with *ij-determinacy*, or constant-depth fixed-arity determinacy, which is more restricted than determinacy.

can convert this into a positive counterexample for PLearn: $l_1 \wedge \dots \wedge l_n \rightarrow q_i$. Similarly, if PLearn asks an entailment membership query on some clause, say, $l_1 \wedge \dots \wedge l_n \rightarrow q_i$, we can turn that into a membership query on the interpretation $l_1, \dots, l_n, \neg q_i$ after substituting a unique skolem constant for each variable in the clause. The answer to the entailment query is *true* iff the answer to the membership query is *false*.

One limitation of our algorithm is that it assumes that the *supported by* relation, \succ , is given. While this is a reasonable assumption in some planning domains, where it is known which goals occur as subgoals of which, it is desirable to learn this relation. Unfortunately, this seems difficult due to a number of problems. One of the main difficulties is that it is sometimes not possible to determine which, of the set of consequents of an antecedent, is the prime consequent. For example, consider the target $\Sigma : \{l_1(x) \wedge l_2(x) \rightarrow l_3(x); l_1(x) \wedge l_3(x) \rightarrow l_4(x)\}$. Given the counterexample $l_1(c) \wedge l_2(c) \rightarrow l_4(c)$, the literal $l_4(c)$ is not a correct consequent, but $l_3(c)$ is. Although Lemma 3 says that prime consequent is a right consequent to choose, without knowing the order it is not clear how to identify it. Learning all possible clauses while maintaining all consequents also does not seem to work, resulting in spurious matches between some of these redundant clauses and counterexamples in some cases.

As shown in (Reddy & Tadepalli, 1997b), Horn programs can be used to express goal-decomposition rules (d-rules) for planning using the situation-calculus formalism. We believe that the algorithm discussed here and its extensions can be applied to learn d-rules, which is an important problem in speedup learning. d-rules are a special case of hierarchical task networks or HTNs (Erol, Hendler, & Nau, 1994)—in that HTNs allow partial ordering over subgoals and non-codesignation constraints over variables whereas d-rules do not. Nevertheless, it can be shown that HTNs can be expressed as Horn programs.

Acknowledgments

We gratefully acknowledge the support of ONR under grant # N00014-95-1-0557 and NSF under grant # IRI-9520243. We thank Roni Khardon and the anonymous reviewers of ICML-98 and ILP-98 for their insightful comments.

References

- Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2, 319–342.
- Arimura, H. (1997). Learning acyclic first-order horn sentences from entailment. In *Proceedings of the Eighth International Workshop on Algorithmic Learning Theory*. Ohmsha/Springer-Verlag.
- Cohen, W. (1995a). Pac-learning non-recursive prolog clauses. *Artificial Intelligence*, 79(1), 1–38.
- Cohen, W. (1995b). Pac-learning recursive logic programs: efficient algorithms. *Jl. of AI Research*, 2, 500–539.
- De Raedt, L. (1997). Logical settings for concept learning. *Artificial Intelligence*, 95(1), 187–201.
- Džeroski, S., Muggleton, S., & Russell, S. (1992). Pac-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 128–135.
- Erol, K., Hendler, J., & Nau, D. (1994). HTN planning: complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press.
- Frazier, M., & Pitt, L. (1993). Learning from entailment: An application to propositional Horn sentences. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 120–127.
- Khardon, R. (1996). Learning to take actions. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 787–792.
- Khardon, R. (1998). Learning first order universal horn expressions. In *Proc. of the Eleventh Annual Conf. on Computational Learning Theory (COLT-98)*.
- Lloyd, J. (1987). *Foundations of Logic Programming* (2nd ed.). Springer-Verlag, Berlin.
- Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pp. 368–381. Ohmsha/Springer-Verlag.
- Natarajan, B. (1989). On learning from exercises. In *Proceedings of the Second Workshop on Computational Learning Theory*, pp. 72–87. Morgan Kaufmann.
- Page, C. D. (1993). *Anti-Unification in Constraint Logics: Foundations and Applications to Learnability in First-Order Logic, to Speed-up Learning, and to Deduction*. Ph.D. thesis, Univ. of Illinois, Urbana, IL.
- Plotkin, G. (1970). A note on inductive generalization. In Meltzer, B., & Michie, D. (Eds.), *Machine Intelligence*, Vol. 5, pp. 153–163. Elsevier North-Holland, New York.
- Reddy, C., & Tadepalli, P. (1997a). Learning goal-decomposition rules using exercises. In *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann.
- Reddy, C., & Tadepalli, P. (1997b). Learning Horn definitions using equivalence and membership queries. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*. Springer Verlag.
- Sammur, C. A., & Banerji, R. (1986). Learning concepts by asking questions. In *Machine learning: An artificial intelligence approach*, Vol. 2. Morgan Kaufmann.

RL-TOPs: An Architecture for Modularity and Re-Use in Reinforcement Learning

Malcolm R. K. Ryan

Department of Artificial Intelligence
School of Computer Science and Engineering
University of New South Wales
Sydney 2052 Australia
malcolmr@cse.unsw.edu.au

Mark D. Pendrith*

Department of Artificial Intelligence
School of Computer Science and Engineering
University of New South Wales
Sydney 2052 Australia
pendrith@cse.unsw.edu.au

Abstract

This paper introduces the **RL-TOPs** architecture for robot learning, a hybrid system combining teleo-reactive planning and reinforcement learning techniques. The aim of this system is to speed up learning by decomposing complex tasks into hierarchies of simple behaviours which can be learnt more easily. Behaviours learnt in this way can subsequently be re-used to solve a variety of problems, reducing the need to learn every new task from scratch. It is even possible to learn multiple behaviours simultaneously, thus making more efficient use of experience. We demonstrate these advantages in a simple simulated environment.

1 INTRODUCTION

Programming robots is difficult (Dorigo, 1996). Often the best way for the robot to solve a problem is unknown, or hard to express. The real world is dynamic, and to be truly autonomous, robots need to be able to cope with a changing environment (Covigaru & Lindsay, 1991). Robot programming would be greatly simplified if robots were able to learn appropriate behaviours of their own accord, and could adapt those behaviours to changes in the world around them. Reinforcement Learning (RL) provides an elegant theoretical framework to achieve these goals but often fails in practice due to the "curse of dimensionality" operating in large state spaces and with complex problems such as those typically found in real robot domains. As

the number of states grows, the problem of determining the best action to perform in each state becomes impossibly difficult.

This problem is not peculiar to RL, traditional robot programmers have faced it also. It is generally not feasible to produce a single monolithic control system which handles all possibilities. Instead, the trend has been towards *behaviour-based programming* (Matarić, 1996). A complex task is decomposed into a set of simple modules or *behaviours*, each of which handle a small part of the problem. These are more easily programmed, and can then be combined to solve the full problem.

One such technique, Brook's *subsumption architecture* (Brooks, 1986), has been successfully transferred to the RL domain, to simplify learning. Mahadevan and Connell (Mahadevan & Connell, 1992) showed that a complex learning task (robot box-pushing), which could not be learnt by a simple reinforcement learner, could, however, be learnt by decomposing it into a subsumption-style hierarchy of simple behaviours, and learning each of these behaviours as distinct reinforcement learning tasks. Thus the robot effectively had several separate learning modules, each of which works independently to learn a sub-part of the task, but which can all cooperate together to provide the overall solution to the problem.

Task decomposition of this kind is well recognised as a way to improve learning rates. As each module only has to learn its behaviour on a small subset of possible states, its search-space is reduced, and so it can find the optimal policy more quickly. Other authors to have produced algorithms based on this realisation include Kaelbling's HDG (Kaelbling, 1993), Dayan and Hinton's Feudal Reinforcement (Dayan & Hinton, 1992) and Dietterich's MAXQ (Dietterich, 1997) algorithms. These algorithms differ from Mahadevan and Connell's

* Current address: Daimler-Benz Research and Technology Center, 1510 Page Mill Rd, Palo Alto, CA 94304, USA. e-mail: pendrith@rtna.daimlerbenz.com

in that they are based on more geometrical decompositions of the world, rather than using specific domain knowledge to define the behaviours. Because of this, they appear to be less applicable to problems in robotics, which involve high-dimensional state information, from a variety of sensing apparatus, without a simple uniform geometry.

The advantages of the subsumption-architecture, however, are offset by the rigidity of the representation used. The hierarchy has to be designed by hand by the programmer, which can be a non-trivial task for many problems. What is more, a new task requires a new set of behaviours and a new hierarchy. Is it possible to design a more flexible system that can automatically build behaviour hierarchies to solve particular problems? Can behaviours learnt to solve one task be re-used to accelerate the learning of others? These are the questions that this paper seeks to address.

2 TELEO-REACTIVE PLANNING

This problem of selecting and ordering an appropriate set of predefined behaviours to achieve a certain goal has traditionally been the domain of planning algorithms. Historically, planning systems have been deemed unsuitable for robot control, because they failed to model the complexity of the real world. Plans were based on sequences of instantaneous actions, which were expected to succeed every time; but in the real world actions take time to perform, and are not always reliable. However modern planning algorithms are now able to produce plans which closely resemble the behaviour based architectures of Brooks and others. Plans can now include durative actions, which operate over a period of time. Execution of plans is reactive (i.e. the state of the world is constantly re-evaluated to determine which action to perform), and universal (i.e. contingencies exist for all situations).

One such planner is Nilsson's Teleo-Reactive (TR) planning system (Nilsson, 1994). It is based around the notion of a *teleo-operator* (or *TOP*), which is a means of describing a durative action in terms of its conditions and effects. A TOP consists of an action a , a pre-image π and a post-condition λ . The pre-image and post-condition are conjunctions of predicates from the planner's state description language. The action may be a simple primitive action, or may be a complex behaviour in its own right. The TOP $a : \pi \rightarrow \lambda$ signifies that if a is executed while π is true, then λ will eventually become true. Until such time as λ is

achieved, π is maintained¹.

Teleo-reactive plans are represented as structures called *TR-Trees*. Nodes in TR-Trees represent state descriptions, with the root node as the goal. Connections between nodes are labelled with actions, indicating that if the action shown is executed in the lower node, then the condition of the upper node will eventually be achieved.

TR-trees are executed reactively. The nodes in the tree are continually re-evaluated and the action corresponding to the shallowest true node is executed. If at any time there is no true node in the tree, then the planner can be reactivated to grow the plan to cover the new situation; thus TR-trees represent (near-) universal plans.

3 REINFORCEMENT LEARNT BEHAVIOURS AND TELEO-OPERATORS

Like TOPs, behaviours acquired by reinforcement learning are also durative actions with a pre-image (application space) and a post-condition (goal). Given a suitable language to describe these attributes, a set of reinforcement learnt behaviours can easily be represented as a list of TOPs. A TR-planner could then be used to combine these behaviours automatically into a hierarchy to solve a given problem, removing the need for the programmer to do this by hand.

Furthermore, the same TOP descriptions can also be used at the lower level as reinforcement schema for the learning algorithm: The post-condition, if achieved, indicates success, which should be rewarded. Prematurely quitting the pre-image indicates failure, which carries a punishment. Thus the one description has two functions: it is used at the high level to tell the planner how to use the behaviour, and at the low level to tell the learner what it is trying to learn. This duality is the basis of the Reinforcement Learnt TOPs (RL-TOPs) system.

4 THE RL-TOPS ARCHITECTURE

The RL-TOPs architecture is a combination of a simple goal-regression TR-planner, and the discounted-

¹A TOP may also have side-effects which are not part of its post-condition, but these are not relevant to the current discussion.

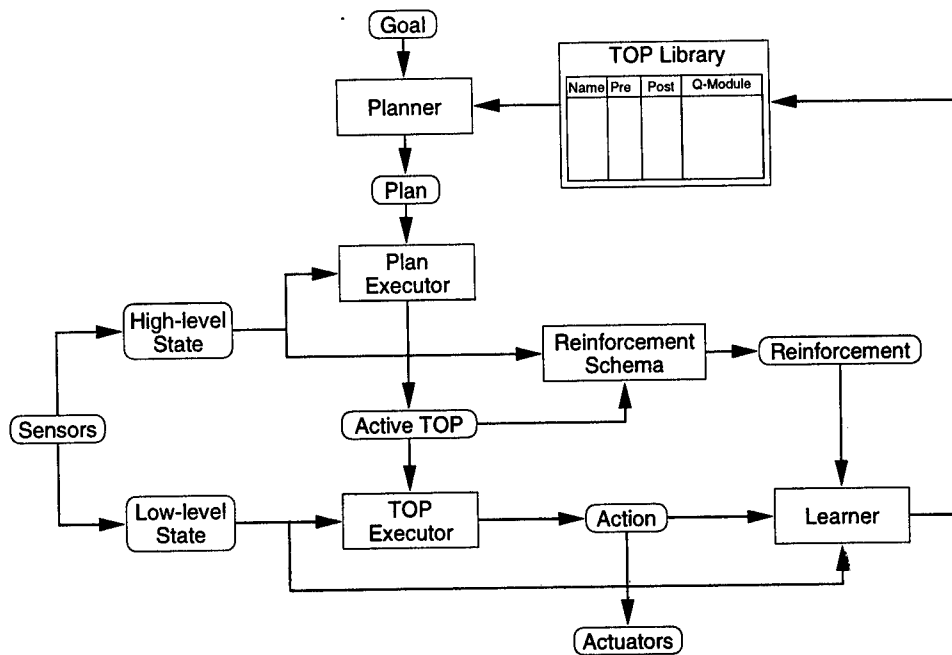


Figure 1: The RL-TOPS architecture.

reward reinforcement learning algorithm C-Trace² (Pendrith & Ryan, 1996). An outline is shown in Figure 1.

Based on the domain and the problem to be solved, the user provides five things:

- A **Low-level State** representation, based on the robot's sensors,
- A set of primitive **Actions**, based on the available actuators,
- A **High-level State** description language (which includes whatever features of the state space are likely to be relevant to the planner, including the goal),
- A **Goal** description,
- A set of behaviour descriptions of the form (*Name*, *Pre-image*, *Post-condition*), which form the **RL-TOP Library**.

The first thing the system does is to supplement each of these RL-TOPs with its own Q-Module. This con-

²The actual reinforcement learning algorithm used is not important, except insofar as it must support learning from both successful and unsuccessful trials. This includes most common RL algorithms such as Q-Learning (Watkins, 1989) and SARSA(λ) (Singh & Sutton, 1996).

tains all the information required by the reinforcement learning algorithm to represent the behaviour. The primary component is the utility (or Q) function, but there may be other components depending on the algorithm. Unless previously saved behaviours are being re-used, the Q-function is initialised to be zero everywhere.

Now, given the goal definition and the library of behaviours available to it, the **Planner** constructs a plan in the form of a TR-Tree. The Planner only constructs as much of the tree as is necessary at any time. Initially the tree consists of just the goal node. As the agent encounters situations which aren't covered by the plan, the Planner will add new nodes to the tree to cover these states, and will add appropriate actions to the plan to link them in to the tree.

The plan is passed to the **Plan Executor**, which also reads the current high-level state description, and chooses which TOP to execute. If the plan does not cover the state, then the Executor re-calls the Planner. Otherwise, the selected TOP is passed to the TOP Executor.

The **TOP Executor** takes the active RL-TOP and the current low-level state, and decides which low-level action to execute. Typically, this will be the policy action provided by the TOP's Q-Module, but an occasional exploratory action may also be performed. For

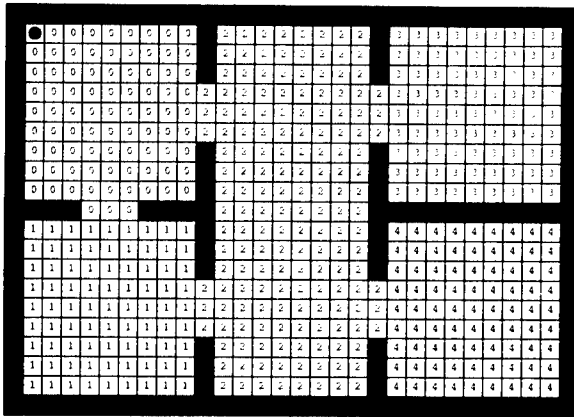


Figure 2: The gridworld domain.

the experiments detailed in this paper, the ϵ -greedy exploration algorithm (Thrun, 1992) was used, with $\epsilon = 0.1$ (i.e. at each step a random exploratory action is chosen with probability 1 in 10.)

The result of the executed action, in terms of changes in the high-level state description, is used by the **Reinforcement Schema** to determine the reinforcement feedback, r , to provide to the Learner. This unit determines whether, in terms of the its pre-image and post-condition, the RL-TOP has succeeded or failed. If the post-condition has become true, then the TOP has succeeded, and a reward of $r = +1$ is returned. Otherwise, if the pre-image is no longer true, then the TOP has failed (by exiting its application space prematurely), and a punishment of $r = -1$ is returned. If neither of these is the case, then $r = 0$.

Combining the low-level state and action information, and the reinforcement signal provided by the Reinforcement Schema, the **Learner** then performs the appropriate update on the RL-TOP's Q-Module, according to whatever reinforcement learning algorithm is used. Then the process repeats, with the Plan Executor deciding which TOP to execute for the next time step, until the goal is achieved.

5 EXPERIMENTAL DOMAIN

Experimental work is currently under way to demonstrate the RL-TOPs architecture on an insectoid robot called Prometheus, aiming to get the robot to learn how to walk towards a beacon. Results from this platform are not yet available, so a simple simulated domain was constructed to demonstrate the system.

The simulation consists of an agent in a 30×21 grid-

RL-TOP	pre-image	post-condition
go02	room(0)	room(2)
go20	room(2)	room(0)
go12	room(1)	room(2)
go21	room(2)	room(1)
go32	room(3)	room(2)
go23	room(2)	room(3)
go42	room(4)	room(2)
go24	room(2)	room(4)

Table 1: RL-TOPs used for gridworld experiments.

world, as shown in Figure 2. At the low-level, the agent can sense its position within the world (as an xy -coordinate) and has four actions available to it, to move north, south, east or west. Each action is guaranteed to succeed unless there is a wall in the way.

The world is divided into five rooms, labelled 0 through 4, and the agent's goal is to reach a particular one, from a randomly chosen starting position. The high-level state and action descriptions are all in terms of which room the agent occupies, given by the predicate $\text{room}(R)$.

For each of the experiments following, the agent was allowed to run for 400 trials, each starting at a random location in the world and finishing when the goal is achieved. The length of each trial, in terms of the total number of low-level actions performed, was recorded. Twenty such runs were performed for each algorithm presented, and the results are the average trial lengths over these twenty runs.

The measurement we are interested in comparing is the time taken to learn the task, that is, the number of primitive actions performed before the agent converged to an optimal (or near-optimal) policy. To this end, the graphs compare cumulative trial lengths for each experiment. The cumulative trial length is the sum of the lengths all trials up to and including the current one.

5.1 EXPERIMENT 1: MODULAR VS. MONOLITHIC

The first experiment demonstrates the improvement in performance of the modular RL-TOPs architecture over a simple monolithic reinforcement learner. The agent's goal is to reach room 4. The monolithic learner has a single Q-Module which covers the entire state space, whereas the modular learner has been provided with eight RL-TOP descriptions, corresponding

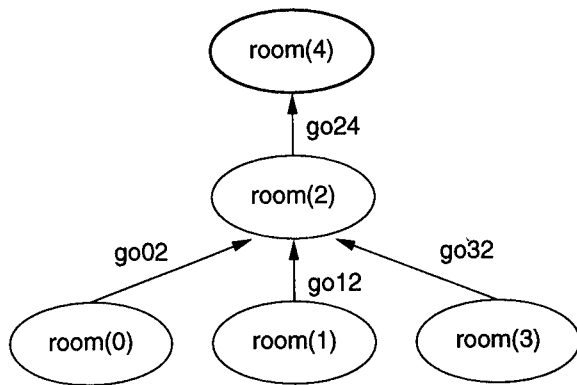


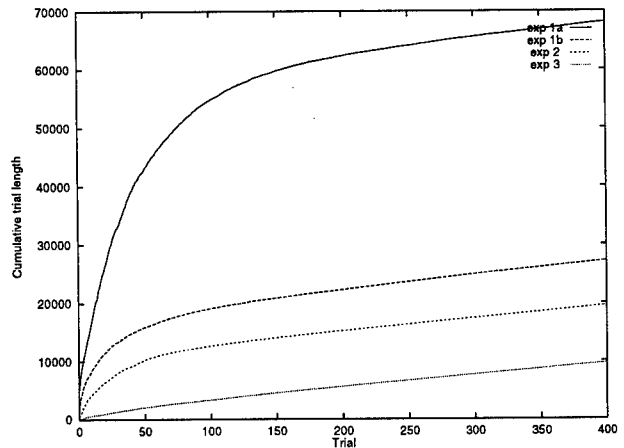
Figure 3: The TR-Tree for going to room 4.

to movement from one room to an adjoining one, as listed in Table 1. The TR-Tree produced by the planner is shown in Figure 3.

The C-Trace learning algorithm was used in both cases, with the learning rate $\beta = 0.1$ and discount factor $\gamma = 0.9$. The monolithic learner was rewarded on success only, with a reinforcement value of 1. As with the RL-TOPs algorithm, the monolithic learner used the ϵ -greedy exploration algorithm, with $\epsilon = 0.1$.

Graph 1 shows the results of the two experiments. Both approaches converged to a nearly optimal policy within about 200 trials, but the monolithic learner took about 40,000 more steps to reach this point. A large part of this difference is established in the first 20 trials, which took the monolithic and modular systems, 25,372 and 11,253 steps respectively. This demonstrates the important difference between the two. In the early stages of learning, when the Q-function is still mostly zero, the only actions that provide any information are those that provide non-zero feedback. Since, in the monolithic case, rewards are few, the agent has nothing to direct it, and a large amount of time is spent aimlessly exploring the world, without learning anything.

In the modular system, however, the application spaces for individual behaviours are smaller, so the rewards (and penalties) are closer at hand. Thus random exploration is more likely to result in useful information more quickly, and learning is significantly faster.



Graph 1: Learning times for gridworld task using (1a) Monolithic learner, (1b) RL-TOPs, (2) RL-TOPs re-using previously learnt behaviours, (3) RL-TOPs using behaviours learnt with concurrent learning.

5.2 EXPERIMENT 2: RE-USING BEHAVIOURS

Another advantage of the modular system over the monolithic is that the individual behaviours learnt in the modular trials can be re-used in a way that the monolithic policy cannot. In the next experiment, the same RL-TOPs from the previous experiment were used, with the Q-Modules saved from each run, in order to solve a new problem.

The goal is now to reach room 3. The new plan is shown in Figure 4. Notice that it includes two of the behaviours learnt in the previous experiment *go02* and *go12*. The other two behaviours, *go42* and *go23*, haven't been used before and still need to be learnt.

From the graph, we can see that a significant amount of time is saved in learning to perform this new task, compared to the previous one, which did not have the benefit of pre-existing behaviours. The reason for this is obvious: the agent does not need to waste time re-learning the *go02* and *go12* behaviours.

Still, a significant amount of time was taken up with learning the *go23* behaviour which would appear to be redundant. Although the agent has never performed this behaviour before, it has nevertheless spent a lot of time in room 2 in the previous experiment, albeit while executing a different behaviour. Common sense suggests that this prior experience should be of some use in learning the new behaviour more quickly. Is it possible to make use of information gathered while executing one behaviour in order to learn another? We

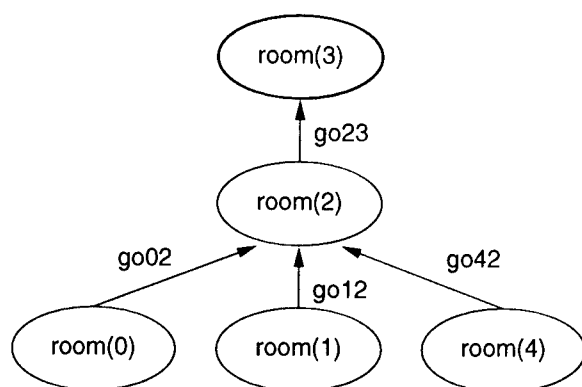


Figure 4: The TR-Tree for going to room 3.

address this question in the next section of this paper.

6 CONCURRENT LEARNING: MAKING BETTER USE OF EXPERIENCE

At this point, one under-appreciated feature of certain RL algorithms comes to our aid. Algorithms such as Q-Learning and C-Trace (but not SARSA) are *off-policy* learners, which means that they sequence of actions presented to the learner do not have to correspond to an actual execution of the policy (Sutton & Barto, 1998). It is even possible to learn one behaviour while executing a quite different one, so long as their application spaces overlap.

This technique, called *concurrent learning* can be added to the RL-TOPs architecture by a simple modification to the Learner module. Rather than just updating the Q-Module of the currently active TOP, the Learner examines the RL-TOP Library and selects all the behaviours which are eligible to be updated. This includes any behaviour the pre-image of which was satisfied before the most recent action was performed. Thus, to use the simulation above as an example, if the agent executes some action in room 2, then, regardless of the result of the action, all those behaviours which have room(2) as their pre-image, will be eligible to be updated.

The Learner then consults the Reinforcement Schema for each behaviour separately, to find out the reinforcement value for that particular TOP. For some, the action just executed may comprise success, for others failure, and for others neither of the two. The Learner uses the reinforcement value for each TOP, to update that TOP's Q-Module. Then execution proceeds as

usual.

This technique should significantly speed up learning more than one task, because it makes more effective use of experience gained.

6.1 EXPERIMENT 3: CONCURRENT LEARNING

To demonstrate the benefit of concurrent learning the two previous experiments were repeated, but this time with all eligible behaviours being learnt concurrently. First the agent did 400 trials with room 4 as its goal. Then, using the same learnt behaviours, the goal was changed to room 3. Graph 1 shows the results of this run. Compare these to the results of experiment 2, which had the same goal, but did not use concurrent learning. The concurrent system converged in very little time at all. The behaviour *go23* was almost completely optimised before it was even run. The only behaviour to be learnt was *go42*, because the agent had had no prior experience with performing any actions in room 4.

7 RELATED WORK

In addition to those already mentioned, other hierarchical learning/planning systems of note include Singh's Compositional Q-Learning system (Singh, 1992), which learns a Q-function for a complex problem by constructing a *gating module* which selects an appropriate lower-level behaviour at each step; and the work of Precup et al. (Precup, Sutton, & Singh, 1997), which extends standard dynamic programming techniques to be able to use macro actions (behaviours) as well as primitive actions in their policies. Both of these systems assume that the behaviours that are used are already fully specified, perhaps by earlier learning runs.

Benson has produced a system that is complementary to that presented here. His TRAIL (Benson, 1996) architecture takes an existing set of actions or behaviours and, by guided experiments, learns appropriate TOP descriptions. It may be possible to combine that work and this, to produce a system in which learnt information goes in both directions, refining both the behaviours and the model.

8 CONCLUSION

As has been demonstrated, modular decomposition is an effective way to improve the speed of reinforce-

ment learning algorithms. The Reinforcement Learnt Teleo-operators (RL-TOPs) architecture, combining low-level reinforcement learning with high-level symbolic planning, is an elegant and effective way of expressing this decomposition. The system allows the automatic construction of appropriate hierarchies of learnt behaviours to solve a given problem, and provides a means of re-using behaviours learnt in one task, for solving another. With the addition of concurrent learning of multiple behaviours, this can greatly improve learning times over a variety of problems.

A limitation of this system is that the policy learnt is sub-optimal because the agent cannot "cut corners" between behaviours. Work is in progress to find a way to allow the agent to benefit from the domain information given by the task decomposition, while still being able to converge eventually to an optimal policy.

Another avenue for future research would be to investigate the question of what to do when the programmer-specified TOPs are insufficient to find a path to the goal. Possibly the system could be extended so as to postulate its own new behaviours in this state. However, this is likely to be a very difficult problem.

Acknowledgements

We would like to gratefully acknowledge the assistance of Christina Cook in developing the ideas contained in this paper.

References

- Benson, S. (1996). *Learning Action Models for Reactive Autonomous Agents*. Ph.D. thesis, Department of Computer Science, Stanford University.
- Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 14-23.
- Covigaru, A. A., & Lindsay, R. K. (1991). Deterministic autonomous systems. *AI Magazine*, 12(3), 110-117.
- Dayan, P., & Hinton, G. E. (1992). Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, 5, 271-278.
- Dietterich, T. G. (1997). Hierarchical reinforcement learning with the MAXQ value function decomposition. Tech. rep., Computer Science Department, Oregon State University.
- Dorigo, M. (1996). Editorial: Introduction to the special issue of learning autonomous robots. *IEEE Transactions on Systems, Man and Cybernetics*, 26(6).
- Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the 10th International Conference on Machine Learning*. Morgan Kaufmann.
- Mahadevan, S., & Connell, J. (1992). Automatic programming of behaviour-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3).
- Matarić, M. J. (1996). Behaviour based control: Examples from navigation, learning and group behaviour. *Journal of Experimental and Theoretical Artificial Intelligence*.
- Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1, 139-158.
- Pendrith, M. D., & Ryan, M. R. K. (1996). Actual return reinforcement learning versus temporal differences: Some theoretical and experimental results. In *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufmann.
- Precup, D., Sutton, R. S., & Singh, S. (1997). Planning with closed-loop macro actions. In *Proceedings of the AAAI Fall Symposium on Model-directed Autonomous Systems*.
- Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3), 323-340.
- Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*.
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.
- Thrun, S. B. (1992). The role of exploration in learning control. In White, D., & Sofge, D. (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold.
- Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Cambridge.

Evolving Structured Programs with Hierarchical Instructions and Skip Nodes

Rafał Sałustowicz Jürgen Schmidhuber

IDSIA

Corso Elvezia 36

6900 Lugano

Switzerland

{rafal, juergen}@idsia.ch

Abstract

To evolve structured programs we introduce H-PIPE, a hierarchical extension of Probabilistic Incremental Program Evolution (PIPE). Structure is induced by “hierarchical instructions” (HIs) limited to top-level, structuring program parts. “Skip nodes” (SNs) allow for switching program parts on and off. They facilitate synthesis of certain structured programs. In our experiments H-PIPE outperforms PIPE: structural bias can speed up program synthesis.

Keywords: Probabilistic Incremental Program Evolution, Structured Programs, Hierarchical Programs, Non-Coding Segments.

1 Introduction

Overview. Automatic program synthesis is of interest because it addresses the problem of searching in general algorithm space as opposed to more limited search spaces like those of, say, feedforward neural networks. Hierarchical Probabilistic Incremental Program Evolution (H-PIPE) is a novel method for synthesizing *structured* programs. It uses the PIPE paradigm (Sałustowicz and Schmidhuber, 1997) to iteratively generate successive populations of functional programs from an adaptive probability distribution over all possible programs constructible from a predefined instruction set. As in PIPE the probability distribution is adapted in three ways: (1) Each iteration the probability of the best program in the current population is increased; (2) occasionally the probability of the best program found so far (elitist) is increased; (3) sometimes probabilities are mutated to better explore

the search space. H-PIPE uses “hierarchical instructions” (HIs) and “skip nodes” (SNs). HIs can be used to combine lower-level program parts, thus inducing structure. SNs function as gates that allow for keeping program parts dormant without losing them in the course of evolution. In combination with HIs they also enable H-PIPE to substitute program parts by superior partial solutions discovered at later evolutionary stages.

Structure. Early genetic programming (GP) work (Dickmanns et al., 1987) as well as Adaptive Levin Search (Schmidhuber, 1997, Schmidhuber et al., 1997b) allow for powerful programs with arbitrary loops etc. Sometimes, however, it is beneficial to introduce inductive bias by appropriately constraining the search space of possible programs. Except for programs evolved by tree-based GP (Cramer, 1985; Koza, 1992), however, not much work has been done on evolution of programs with significant *structural* constraints. There are two such GP variants.

The first reuses program parts, usually in a way less general than that achievable through arbitrary jumps (Dickmanns et al., 1987). Typically subprograms are generated and/or extracted from evolved programs; they may then be called in a usually non-recursive fashion from different positions in the code. Examples are: “automatically defined functions” and encapsulation (Koza, 1992), module acquisition (Angeline and Pollack, 1992), adaptive representations through learning (Rosca and Ballard, 1996), automatically defined macros (Spector, 1996). Other approaches do not generate or extract subprograms but restrict GP’s recombination operator such that it cannot destroy certain program parts to be reused in the future (e.g., Langdon, 1995; Pringle, 1995; Zannoni and Reynolds, 1997).

The second variant uses grammars to induce struc-

ture, constrain the search space, and provide initial bias to speed up evolution. Examples are context-free (Whigham, 1995, Gruau, 1996) or logic grammars (Wong and Leung, 1996).

Hierarchical Instructions. H-PIPE's programs are composed of instructions from a fixed instruction set $S = \{I_1, I_2, \dots, I_z\}$. Each node of the code tree contains an instruction I and can have several son nodes whose instructions are viewed as arguments of I . Programs with hierarchical instructions (HIs) are special cases of programs constrained by context-free grammars: We partition S into m disjoint, non-empty instruction sets S^0, S^1, \dots, S^m , and ensure that all "terminal instructions" – instructions with zero arguments – are in S^0 . Hierarchical order is imposed as follows: Each argument of an instruction in S^v is in S^v or in the "lower level" set S^{v-1} . At least one argument must be in S^{v-1} , except when $v = 0$. Higher-level instructions can be used to combine program parts made out of lower-level instructions, thus inducing structure.

Non-Coding Program Parts. Non-coding program parts ("introns") are those that do not affect the results the program calculates. E.g., in $f(x) = x * 1$, the $"*1"$ part is non-coding. Most previous work on non-coding program parts focuses on genetic program synthesis (Blickle and Thiele, 1994, McPhee and Miller, 1995, Nordin et al., 1996, Haynes, 1996, Wineberg and Oppacher, 1996). Usually non-coding program parts evolve or can be inserted to protect coding program parts (parts that *do* affect results calculated by the program) from destructive genetic recombination operators (Blickle and Thiele, 1994, McPhee and Miller, 1995, Nordin et al., 1996, Haynes, 1996). Blickle and Thiele (1994), as well as McPhee and Miller (1995), however, point out that large blocks of non-coding segments in tree-based GP programs cause very slow convergence and difficulties in escaping from local minima. Haynes (1996), on the other hand, shows that artificial removal of non-coding segments from those programs leads to premature convergence. Nordin, Francone, and Banzhaf (1996) investigate the role of non-coding segments in a GP approach based on variable-length strings. They note that non-coding segments may play an important role in finding good solutions and speeding up convergence. Wineberg and Oppacher (1996) use *fixed*-length strings and find that non-coding segments reduce the search space and speed up evolution.

General observation. The literature above suggests: in tree-based GP programs with little structure, the effect of non-coding segments is twofold. On the one hand they seem necessary to protect blocks of coding

segments, on the other hand they can hinder discovery of acceptable solutions. In the case of *structured* programs, however, non-coding program parts can both speed up convergence *and* aid in finding good solutions. Loosely speaking, the more structured the programs (e.g., the greater the restrictions on the coding strings), the higher the potential significance of non-coding segments. Our own experiments with skip nodes will add more empirical evidence in this direction.

Skip Nodes (SNs). Much like certain "jump" instructions, skip nodes (SNs) are instructions that allow for skipping program parts. In the context of tree-based functional programs, SNs are functions with n arguments, where n denotes the maximal number of arguments of functions in S . SNs return exactly one of their arguments and ignore the others, which thus represent non-coding program parts if $n > 1$. We will demonstrate the benefits of SNs in structuring parts of H-PIPE programs.

Outline. Section 2 describes the H-PIPE approach. Section 3 compares the use of HIs and SNs to standard PIPE on function regression and 6-bit parity. Section 4 concludes.

2 Hierarchical PIPE

Overview. We will describe H-PIPE, a hierarchical extension of PIPE (Safustowicz and Schmidhuber, 1997). Like PIPE, H-PIPE combines probability vector coding of program instructions (Schmidhuber et al., 1997a, 1997b), Population-Based Incremental Learning (PBIL – Baluja & Caruana, 1995), and tree-coded programs like those used in variants of GP. Unlike PIPE, H-PIPE uses HIs to evolve structured programs and SNs to facilitate this process. We will first describe HIs and then SNs.

2.1 Hierarchical Instructions (HIs)

Program Instructions. H-PIPE's programs are composed from z instructions in the instruction set $S = \{I_1, I_2, \dots, I_z\}$. Each instruction I_j ($1 \leq j \leq z$) is either a function or a terminal. Functions and terminals differ in that the former have one or more arguments and the latter have zero. Thus $S = F \cup T$, where $F = \{f_1, f_2, \dots, f_k\}$ is a function set with k functions and $T = \{t_1, t_2, \dots, t_l\}$ is a terminal set with l terminals. Since $F \cap T = \{\}$, $z = k + l$ holds. Programs are encoded in trees. Each node of the code tree contains an instruction I and can have several

son nodes whose instructions are viewed as arguments of I . To allow for HIs we partition S into m disjoint, non-empty instruction sets S^0, S^1, \dots, S^m , and ensure that all “terminal instructions” – instructions with zero arguments – are in S^0 . Hierarchical order arises as follows: Each argument of an instruction in S^v is in S^v or in the “lower level” set S^{v-1} . At least one argument must be in S^{v-1} , except when $v = 0$. To allow for enforcing descents in the instruction set hierarchy we add “level down” instructions \downarrow_i to all instruction sets S^v ($0 < v \leq m$), where $0 \leq i \leq l(v)$ is the argument index of an instruction $I \in S^v$ with $l(v)$ arguments from S^{v-1} . Although “level downs” take a single argument and return it, they are treated as terminal symbols. Thus each instruction set S^v ($0 < v \leq m$) can be written as $F^v \cup T^v$, where $F^v = \{f_1^v, f_2^v, \dots, f_{k(v)}^v\}$ is a function set with $k(v)$ functions and $T^v = \{\downarrow_0, \downarrow_1, \dots, \downarrow_{l(v)}\}$ is a terminal set containing $l(v)$ “level down” instructions. We also have $S^0 = F^0 \cup T^0$, where $F^0 = \{f_1^0, f_2^0, \dots, f_{k(0)}^0\}$ is a function set with $k(0)$ functions and $T^0 = T$ is a terminal set containing all terminals of S ($l(0) = l$).

To solve a one-dimensional function approximation task one might use $F = \{+, -, *, \%, \sin, \cos, \exp, rlog\}$ and $T = \{x, R\}$, where $\%$ denotes protected division ($\forall y, u \in \mathbb{R}, u \neq 0: y\%u = y/u$ and $y\%0 = 1$); $rlog$ denotes protected logarithm ($\forall y \in \mathbb{R}, y \neq 0: rlog(y) = \log(\text{abs}(y))$ and $rlog(0) = 0$); x is an input variable; and R is a *generic random constant* in $[0;1)$ (see below). To structure this function approximation task as a linear combination of non-linear parts we split the instruction set $S = \{+, -, *, \%, \sin, \cos, \exp, rlog, x, R\}$ into $S^0 = \{*, \%, \sin, \cos, \exp, rlog, x, R\}$ and $S^1 = \{+, -\}$. We then add a \downarrow_0 instruction to S^1 and obtain $S^1 = \{+, -, \downarrow_0\}$. Function and terminal sets for the lower and upper level then become $F^0 = \{*, \%, \sin, \cos, \exp, rlog\}$, $T^0 = \{x, R\}$ and $F^1 = \{+, -\}$, $T^1 = \{\downarrow_0\}$, respectively. Figure 1 shows an example program.

Generic Random Constants. A generic random constant (GRC) (compare also “ephemeral random constant” (Koza, 1992)) is a zero argument function (a terminal). When accessed during program creation, it is either instantiated to a random value from a predefined, problem-dependent set of constants or a value previously stored together with the probability distribution (see below).

Program Representation. With HIs the arity $n(v)$ of a program tree may vary depending on the hierarchical level v . On each level v , $n(v)$ is the maximal number of function arguments required by functions

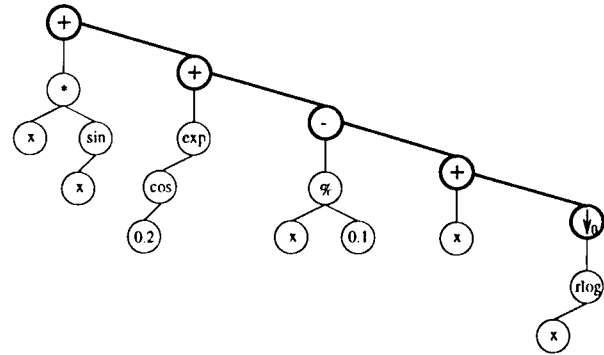


Figure 1: $f(x) = x * \sin(x) + \exp(\cos(0.2)) + x \% 0.1 - (x + rlog(x))$. Exemplary program tree for function approximation constrained to a linear combination of non-linear parts. Top-level structuring instructions from S^1 appear in boldface.

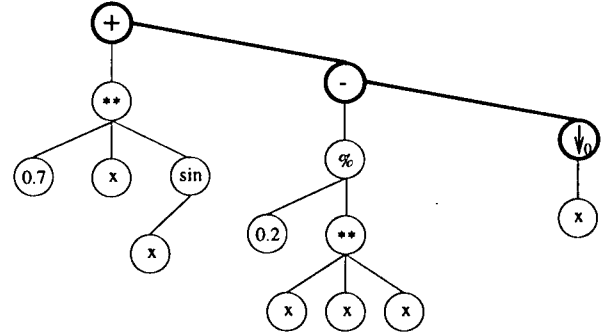


Figure 2: $f(x) = 0.7 * x * \sin(x) + 0.2 \% (x * x * x) - x$. Exemplary program tree for function approximation, with different level-dependent arities. Top-level program parts are 2-ary. Lower level program parts are 3-ary.

in S^v . For instance, in the function approximation example above, if we add to S^0 a three argument function, e.g. $**$, where $**(a_1, a_2, a_3) = a_1 * a_2 * a_3$, then the lower-level part of the program tree will be 3-ary while the top-level part will remain 2-ary, as depicted in Figure 2.

Probability Distribution. The probability distribution is stored in a “hierarchical probabilistic prototype tree” ($H\text{-}PPT$). At each hierarchical level v ($0 \leq v \leq m$) the $H\text{-}PPT$ generally contains infinite $n(v)$ -ary subtrees $PPT^{dw(v)}$, where the list $dw(v) = ((d_{v+1}, w_{v+1}), (d_{v+2}, w_{v+2}), \dots, (d_m, w_m))$ describes the absolute position of a subtree: it contains 0 to $m - 1$ components depending on the hierarchical position v of $PPT^{dw(v)}$ (0 components, if $v = m$). Each component pair (d_i, w_i) describes the position of a higher level node $N_{d_i, w_i}^{dw(i)}$ in $PPT^{dw(i)}$ to which $PPT^{dw(v)}$ is attached. The position of a node

$N_{d_i, w_i}^{dw(i)}$ inside a subtree $PPT^{dw(i)}$ is defined by $N_{d_i, w_i}^{dw(i)}$'s depth $d_i > 0$ ($PPT^{dw(i)}$'s root node has $d_i = 0$) and its horizontal position w_i when subtree nodes with equal depth are read from left to right ($0 \leq w_i < n(i)^{d_i}$). Each node $N_{d_v, w_v}^{dw(v)}$ contains a variable probability vector $\vec{P}_{d_v, w_v}^{dw(v)}$. In addition, each node $N_{d_0, w_0}^{dw(0)}$ contains a random constant $R_{d_0, w_0}^{dw(0)}$. The probability vectors $\vec{P}_{d_v, w_v}^{dw(v)}$, $\forall v : 0 \leq v \leq m$ have $k(v) + l(v)$ components. Each component $P_{d_v, w_v}^{dw(v)}(I)$, $\forall v : 0 \leq v \leq m$ denotes the probability of choosing instruction $I \in S^v$ at $N_{d_v, w_v}^{dw(v)}$. We maintain $\sum_{I \in S^v} P_{d_v, w_v}^{dw(v)}(I) = 1$.

H-PPT Initialization. Each *H-PPT* node $N_{d_v, w_v}^{dw(v)}$ requires an initial probability $P_{d_v, w_v}^{dw(v)}(I)$ for each instruction $I \in S^v$. Furthermore, each bottom level ($v = 0$) node $N_{d_0, w_0}^{dw(0)}$ requires an initial random constant $R_{d_0, w_0}^{dw(0)}$. We pick $R_{d_0, w_0}^{dw(0)}$ uniformly random in the interval $[0;1]$. To initialize instruction probabilities we use for each hierarchical level v a constant probability P_{T^v} for selecting an instruction from T^v and $(1 - P_{T^v})$ for selecting an instruction from F^v . $\vec{P}_{d_v, w_v}^{dw(v)}$ is then initialized as follows:

$$P_{d_v, w_v}^{dw(v)}(I) := \frac{P_{T^v}}{l(v)}, \quad \forall I : I \in T^v \quad \text{and}$$

$$P_{d_v, w_v}^{dw(v)}(I) := \frac{1 - P_{T^v}}{k(v)}, \quad \forall I : I \in F^v$$

Program Generation. Program generation in *H-PIPE* is analogous to program generation in *PIPE* (see Sałustowicz and Schmidhuber, 1997), except that instructions are selected from the appropriate S^v , depending on the hierarchical level. To generate a program *PROG* from *H-PPT*, an instruction $I \in S^v$ is selected with probability $P_{d_v, w_v}^{dw(v)}(I)$ for each accessed node $N_{d_v, w_v}^{dw(v)}$ of *H-PPT*. This instruction is denoted by $I_{d_v, w_v}^{dw(v)}$. Nodes are accessed in a depth-first way, starting at the root node $N_{0,0}$, and traversing *H-PPT* from left to right. Figure 3 shows a *H-PPT* and a corresponding possible program.

Tree Shaping. To reduce memory requirements and allow for discarding elements of the probability distribution that have become irrelevant over time the *H-PPT* is incrementally grown and pruned just like *PIPE*'s probability tree (see Sałustowicz and Schmidhuber, 1997).

Update Rules. *H-PIPE*'s update rules are analogous to *PIPE*'s (see Sałustowicz and Schmidhuber, 1997).

The only difference is the more sophisticated indexing method due to *H-PPT*'s hierarchical structure.

2.2 Skip Nodes (SNs)

Overview. Skip nodes are functions that serve to switch code parts on and off. We will first define SNs for *PIPE*, then for *H-PIPE*.

SNs for PIPE. *PIPE*'s probability distribution is stored in a probabilistic prototype tree (*PPT* – see Sałustowicz and Schmidhuber (1997) for details). Let n denote the maximal arity of the *PPT* (the maximal number of arguments of functions that are not SNs). There are at most n SNs. The i -th is denoted \rightarrow_i . It is a function with n arguments and returns the i -th. Its interpretation is: evaluate the i -th argument but ignore the others.

SNs are elements of the function set F . For instance, if we add SNs to the instruction set of the function approximation example from Section 2.1 we obtain: $F = \{+, -, *, \%, \sin, \cos, \exp, rlog, \rightarrow_0, \rightarrow_1\}$ and $T = \{x, R\}$. Figure 4 shows an unstructured *PIPE* program with SNs. The dashed parts of the program can be

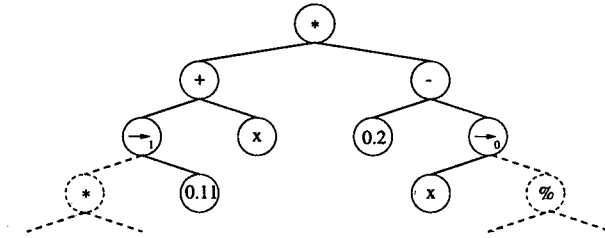


Figure 4: A *PIPE* program with SNs for function approximation: $f(x) = (0.11 + x) * (0.2 - x)$. The dashed parts of the program are non-coding segments.

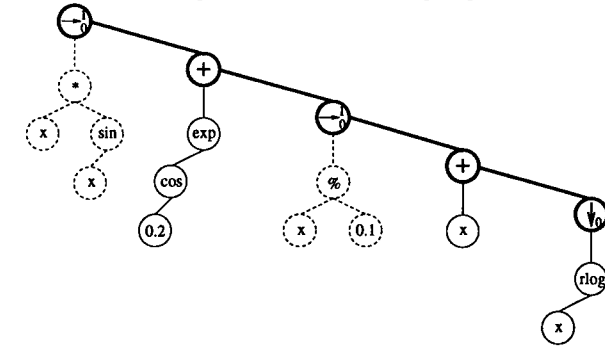


Figure 5: A *H-PIPE* program with SNs for function approximation: $f(x) = \exp(\cos(0.2)) + x + rlog(x)$. The dashed parts of the program are non-coding segments.

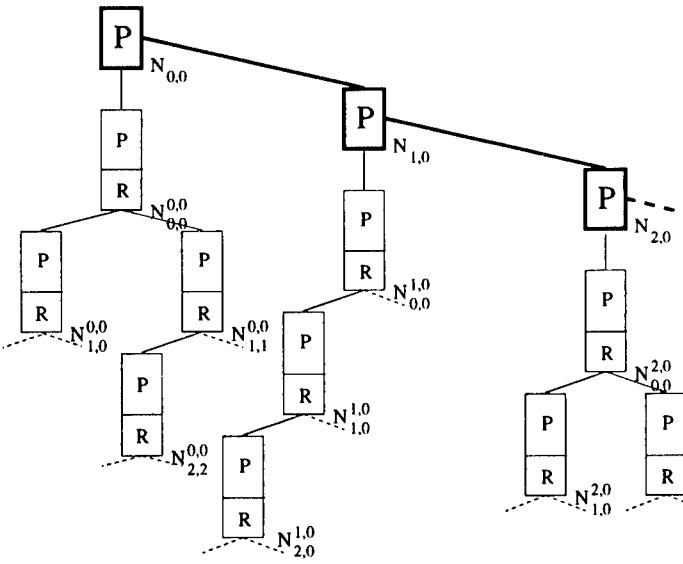


Figure 3: A *H-PPT* (left) and a corresponding possible program (right). The structuring parts of the program are highlighted.

viewed as non-coding segments. Note that they need not even be created during program generation and are therefore computationally cheap.

SNs for H-PIPE. Let $h(v)$ denote the maximal number of arguments of non-SN functions in S^v . At level v ($0 < v \leq m$) there are at most $h(v)$ SNs. The i -th is denoted \rightarrow_i^v . It is a function with $h(v)$ arguments and returns the i -th. Its interpretation is: evaluate the i -th argument but ignore the others. There are no SNs in S^0 .

SNs are elements of the function set F^v . For instance, if we add SNs to the instruction set of the function approximation example from Section 2.1 we obtain: $F^0 = \{*, \%, \sin, \cos, \exp, rlog\}$, $T^0 = \{x, R\}$ and $F^1 = \{+, -, \rightarrow_0\}$, $T^1 = \{\downarrow_0\}$. Figure 5 shows a H-PIPE program with SNs.

Changes to PIPE's and H-PIPE's Update Rules.

(1) Parts of *PPT* or *H-PPT* corresponding to non-coding segments are *not* updated. (2) To mutate probabilities we calculate program size $|\text{PROG}_b|$. With SNs $|\text{PROG}_b|$ denotes the number of nodes in program PROG_b *without* the non-coding segments created by SNs. See Sałustowicz and Schmidhuber (1997) for details.

3 Experiments

To evaluate the impact of HIs and SNs we cross-compare: (1) PIPE, (2) H-PIPE without SNs (H-PIPE-NO-SN), (3) PIPE with SNs (PIPE-SN), (4) and H-PIPE (PIPE with HIs and SNs in the structuring program parts). To illustrate the significance of *appropriate* initial bias we also test H-PIPE with different structuring instructions (H-PIPE-DIFF). We consider a nontrivial continuous function regression problem and the 6-bit parity problem, a discrete task involving just 65 distinct fitness values. For each combination of learning algorithm and problem we conduct 50-200 independent runs to obtain statistically significant results.

3.1 Function Regression

The function to be approximated is plotted in Figure 6. The training data set D_{tr} samples f at 101 equidistant points in the interval $[0;10]$. D_{tr} is used to calculate fitness values during program evolution. Thus, the fitness value of each program PROG is $FIT(\text{PROG}) = \sum_{x \in D_{tr}} |f(x) - \text{PROG}(x)|$, where $\text{PROG}(x)$ denotes the result of applying PROG to data x .

Set-up. We time-constrain all runs to $PE = 100,000$ and use the following parameter setting empirically found to work well: $P_T = P_{T_0} = P_{T_1} = 0.8$, $\varepsilon = 0.000001$, $P_{el} = 0.01$, $PS = 10$, $lr = 0.01$, $P_M = 0.4$, $mr = 0.4$, $T_R = 0.3$, $T_P = 0.999999$, $FIT_s = 0$ (see Salustowicz and Schmid-

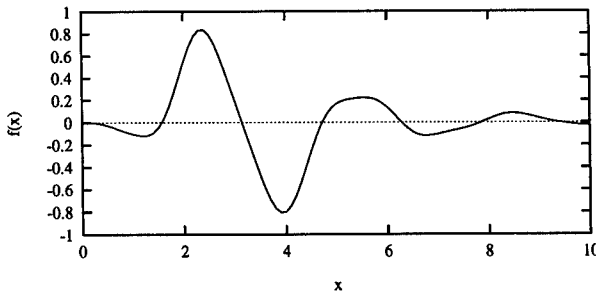


Figure 6: $f(x) = x^3 \cdot e^{-x} \cdot \cos(x) \cdot \sin(x) \cdot (\sin^2(x) \cdot \cos(x) - 1)$

huber (1997) for detailed description of parameters). We use the following instruction sets: (1) PIPE: $F = \{+, -, *, \%, \sin, \cos, \exp, rlog\}$, $T = \{x, R\}$; (2) H-PIPE-NO-SN: $F^1 = \{+, -\}$, $T^1 = \{\downarrow_0\}$, $F^0 = \{*, \%, \sin, \cos, \exp, rlog\}$, $T^0 = \{x, R\}$; (3) PIPE-SN: $F = \{+, -, *, \%, \sin, \cos, \exp, rlog, \rightarrow_0, \rightarrow_1\}$, $T = \{x, R\}$; (4) H-PIPE: $F^1 = \{+, -, \rightarrow_0^1\}$, $T^1 = \{\downarrow_0\}$, $F^0 = \{*, \%, \sin, \cos, \exp, rlog\}$, $T^0 = \{x, R\}$; (5) H-PIPE-DIFF: $F^1 = \{*, \%, \rightarrow_0^1\}$, $T^1 = \{\downarrow_0\}$, $F^0 = \{+, -, \sin, \cos, \exp, rlog\}$, $T^0 = \{x, R\}$.

Results. Figure 7 summarizes all results in form of cumulative histograms. We plot performance u against percentage of programs with $FIT(\text{PROG}) \leq u$. Each point indicates the number of programs with $FIT(\text{PROG})$ equal to or better than its x-axis value: algorithms with better performance have more points with smaller x-values.

PIPE vs. H-PIPE. H-PIPE outperforms PIPE. H-PIPE's fitness in the median run is $FIT_{med} = 2.39$, slightly better than PIPE's with $FIT_{med} = 2.55$. In 82% of all runs H-PIPE finds programs with fitness below 4, while only 67% of all PIPE runs accomplish this. On the other hand, the worst 3% of all H-PIPE runs resulted in programs worse than the best found by all PIPE runs. The median of H-PIPE's program size ($Node_{med} = 92$ nodes) is significantly smaller than PIPE's ($Node_{med} = 157$).

How much of the performance improvement can be attributed to HIs, how much to SNs? To study this question we now compare PIPE and H-PIPE to PIPE with SNs (PIPE-SN) and H-PIPE without SNs (H-PIPE-NO-SN).

PIPE & H-PIPE vs. PIPE-SN. PIPE-SN performs much like PIPE, and worse than H-PIPE. PIPE-SN's $FIT_{med} = 2.70$ is slightly higher than PIPE's ($FIT_{med} = 2.55$). Like PIPE, in 67% of all runs PIPE-

SN found programs with fitness below 4. Its worst programs are slightly better than the worst program among the best of the individual PIPE runs. PIPE-SN's programs ($Node_{med} = 117$) tend to be smaller than PIPE's ($Node_{med} = 157$), but larger than H-PIPE's ($Node_{med} = 92$).

We observe that SNs in unstructured PIPE programs are neither harmful nor beneficial.

PIPE & H-PIPE vs. H-PIPE-NO-SN. H-PIPE-NO-SN is the best competitor, slightly better than H-PIPE, much better than PIPE. H-PIPE-NO-SN's $FIT_{med} = 2.38$ is roughly as good as H-PIPE's $FIT_{med} = 2.39$. In 91% of all runs, however, H-PIPE-NO-SN found programs with fitness below 4, compared to H-PIPE's 82% and PIPE's 67%. Furthermore, unlike with H-PIPE and PIPE, no program found by H-PIPE-NO-SN has fitness above 7.39. The median size of H-PIPE-NO-SN programs, $Node_{med} = 96$, is roughly the same as H-PIPE's ($Node_{med} = 92$) and significantly smaller than PIPE's ($Node_{med} = 157$).

We observe that HIs by themselves increase PIPE's performance. Later (in Section 3.2) we will see that both HIs and SNs are sometimes needed to solve certain tasks more efficiently. But first we will illustrate the importance of choosing the right HIs.

PIPE & H-PIPE vs. H-PIPE-DIFF. H-PIPE-DIFF performs significantly worse than H-PIPE and PIPE. The fitness of the best program found by H-PIPE-DIFF in 50 independent runs is only 7.52. H-PIPE-DIFF's median fitness $FIT_{med} = 10.62$. Compare H-PIPE's and PIPE's, which are 2.39 and 2.55, respectively.

This demonstrates, not unexpectedly, that appropriate initial bias due to "good" HIs is crucial to H-PIPE's success.

Conclusion. HIs can increase PIPE's performance significantly. They need to be selected carefully, however. SNs do not contribute much to solving the function regression task. In case of PIPE they reduce program size without affecting solution quality. In case of H-PIPE they have a slightly detrimental effect on overall performance.

The next experiment will show that for some tasks only the combination of HIs and SNs leads to significant performance improvement.

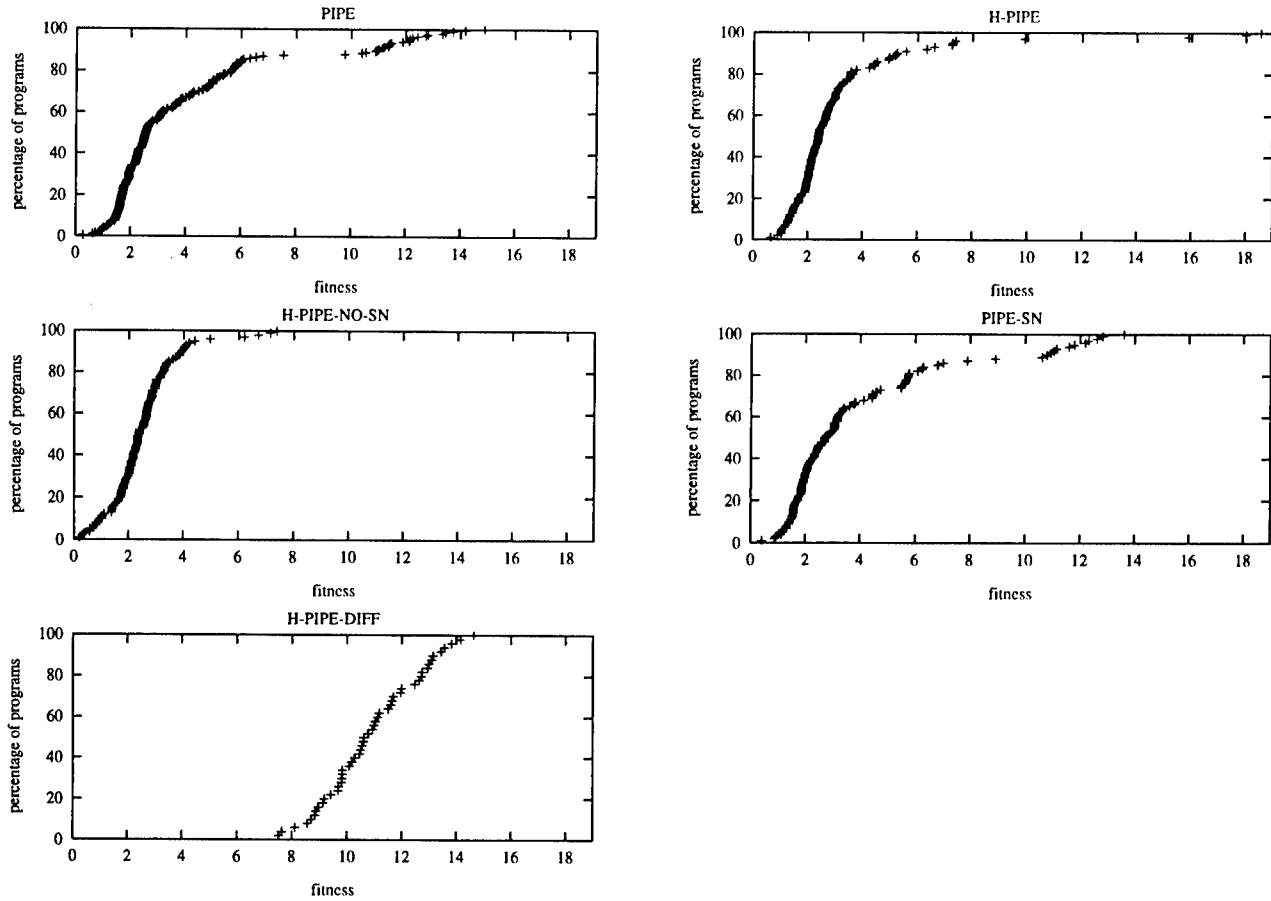


Figure 7: Results for the regression problem.

3.2 6-Bit Parity

The 6-bit parity function has six Boolean arguments represented by integers: 1 for true and 0 for false. It returns 1 if the number of nonzero arguments is odd and 0 otherwise. The fitness of a program is the number of patterns it classifies incorrectly. Best (worst) fitness for classifying all (no) patterns correctly is 0 (64). We use all 64 patterns for training.

Set-up. We time-constrain all runs to $PE = 500,000$ and use the following parameter settings empirically found to work well: $P_T = P_{T^0} = P_{T^1} = 0.6$, $\varepsilon = 0.000001$, $P_{el} = 0.01$, $PS = 10$, $lr = 0.01$, $P_M = 0.4$, $mr = 0.4$, $T_R = 0.3$, $T_P = 0.999999$, $FIT_s = 0$ (see Sałustowicz and Schmidhuber (1997) for detailed description of parameters). Note that, except for P_T , P_{T^0} , and P_{T^1} , all parameters are set to the same values as for the function regression task (see Section 3.1). Most of PIPE's and H-PIPE's parameters seem robust with respect to changing tasks. We use the following instruction sets: (1) PIPE: $F = \{+, -, *, \%, \sin, \cos, \exp, rlog\}$,

$T = \{x_0, x_1, x_2, x_3, x_4, x_5, R\}$; (2) H-PIPE-NO-SN: $F^1 = \{*, \%\}$, $T^1 = \{\downarrow_0\}$, $F^0 = \{+, -, \sin, \cos, \exp, rlog\}$, $T^0 = \{x_0, x_1, x_2, x_3, x_4, x_5, R\}$; (3) PIPE-SN: $F = \{+, -, *, \%, \sin, \cos, \exp, rlog, \rightarrow_0, \rightarrow_1\}$, $T = \{x_0, x_1, x_2, x_3, x_4, x_5, R\}$; (4) H-PIPE: $F^1 = \{*, \%, \rightarrow_0^1\}$, $T^1 = \{\downarrow_0\}$, $F^0 = \{+, -, \sin, \cos, \exp, rlog\}$, $T^0 = \{x_0, x_1, x_2, x_3, x_4, x_5, R\}$; (5) H-PIPE-DIFF: $F^1 = \{+, -, \rightarrow_0^1\}$, $T^1 = \{\downarrow_0\}$, $F^0 = \{*, \%, \sin, \cos, \exp, rlog\}$, $T^0 = \{x_0, x_1, x_2, x_3, x_4, x_5, R\}$. To fit the Boolean nature of the problem the real-valued output of a program is mapped to 0 if negative and to 1 otherwise.

Results. Table 1 summarizes all results. The first column displays for each algorithm the percentage of independent runs leading to perfect solutions within the given time frame (PE). The next three columns show the numbers of program evaluations necessary to find perfect solutions in the shortest, median, and longest run, respectively. The final three columns list the minimal, median, and maximal program sizes embodying perfect solutions.

Table 1: Summary of 6-bit parity results. Best values are in boldface.

Algorithm	6-bit parity				
	solved	Program Evaluations			Nodes
		min	med	max	min-med-max
H-PIPE	94 %	5,700	37,460	397,000	23- 61 -96
PIPE	79 %	3,520	79,950	497,220	24- 64 -137
PIPE-SN	76 %	1,676	73,720	487,930	25- 58 -110
H-PIPE-NO-SN	66 %	3,720	166,740	468,950	21- 49 -85
H-PIPE-DIFF	28 %	38,300	216,570	457,330	24- 61 -94

Comparison. H-PIPE performs best. It solves the task more often and significantly faster (with less program evaluations) than PIPE, PIPE with SNs, and H-PIPE without SNs. PIPE and PIPE-SN have roughly the same performance. PIPE-SN finds slightly fewer solutions, but is faster than PIPE in the median run. The median size of its solutions is also slightly smaller than PIPE's. Although its solution size is smallest in the median run, H-PIPE-NO-SN performs significantly worse than PIPE and PIPE-SN. It finds fewer solutions and requires more than twice as many program evaluations (in the median run). H-PIPE-DIFF with wrong initial bias is worst of all. It needs more than five times as many program evaluations as H-PIPE to find roughly three times fewer solutions.

Conclusion. With this particular task H-PIPE outperforms PIPE. Neither SNs by themselves nor HIs by themselves are able to improve PIPE's performance. In absence of structure SNs' effects are neither harmful nor beneficial, while HIs by themselves decrease PIPE's performance. The combination of both HIs (embodying the proper initial bias) and SNs in H-PIPE, however, allows for significant improvement.

4 Conclusion

H-PIPE, a novel method for synthesizing *structured* programs, uses hierarchical instructions (HIs) to structure programs and skip nodes (SNs) to facilitate their synthesis. HIs combine program parts, while SNs allow for non-coding segments. In our experiments, HIs by themselves sometimes worked extremely well, but not always. Then, however, combining them with SNs helped to achieve dramatic improvement. SNs by themselves were useless for improving performance. Our review of previous work on non-coding segments suggests that non-coding segments seem to require *structured* code to unfold their benefits. Our own results add further empirical evidence in this vein.

Limitations and Future Work. HIs are chosen a priori — currently there is no recipe for finding the optimal ones. But it may be possible to automatize the HI selection process itself by making it subject to data-driven evolutionary optimization.

Acknowledgments

Thanks to Marco Wiering, Nicol Schraudolph, and Jieyu Zhao for valuable comments and suggestions that helped to improve a draft of this paper. This work was supported by SNF grant 2100-49'144.96 "Long Short-Term Memory".

References

- Angeline, P. J. and Pollack, J. B. (1992). The evolutionary induction of subroutines. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, pages 236–241, Hillsdale, NJ. Lawrence Erlbaum Associates.
- Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 38–46. Morgan Kaufmann Publishers, San Francisco, CA.
- Blickle, T. and Thiele, L. (1994). Genetic programming and redundancy. In Hopf, J., editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany. Max-Planck-Institut für Informatik (MPI-I-94-241).
- Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In Grefenstette, J., editor, *Proceedings of an International Conference on Genetic Algorithms and*

- Their Applications*, pages 183–187, Hillsdale, NJ. Lawrence Erlbaum Associates.
- Dickmanns, D., Schmidhuber, J., and Winklhofer, A. (1987). Der genetische Algorithmus: Eine Implementierung in Prolog. Fortgeschrittenenpraktikum, Institut für Informatik, Lehrstuhl Prof. Radig, Technische Universität München.
- Gruau, F. (1996). On using syntactic constraints with genetic programming. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, USA.
- Haynes, T. (1996). Duplication of coding segments in genetic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 344–349, Portland, OR.
- Koza, J. R. (1992). *Genetic Programming – On the Programming of Computers by Means of Natural Selection*. MIT Press.
- Langdon, W. B. (1995). Directed crossover within genetic programming. Research Note RN/95/71, University College London, Gower Street, London WC1E 6BT, UK.
- McPhee, N. F. and Miller, J. D. (1995). Accurate replication in genetic programming. In Eshelman, L., editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA. Morgan Kaufmann.
- Nordin, P., Francone, F., and Banzhaf, W. (1996). Explicitly defined introns and destructive crossover in genetic programming. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 6, pages 111–134. MIT Press, Cambridge, MA, USA.
- Pringle, W. R. (1995). ESP: Evolutionary structured programming. Technical report, Penn State University, Great Valley Campus, PA, USA.
- Rosca, J. P. and Ballard, D. H. (1996). Discovery of subroutines in genetic programming. In Angeline, P. and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, page Chapter 9. MIT Press, Cambridge, MA.
- Salustowicz, R. P. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141.
- Schmidhuber, J. (1997). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873.
- Schmidhuber, J., Zhao, J., and Schraudolph, N. (1997a). Reinforcement learning with self-modifying policies. In Thrun, S. and Pratt, L., editors, *Learning to learn*, pages 293–309. Kluwer.
- Schmidhuber, J., Zhao, J., and Wiering, M. (1997b). Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28:105–130.
- Spector, L. (1996). Simultaneous evolution of programs and their control structures. In Angeline, P. and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, page Chapter 7. MIT Press, Cambridge, MA, USA.
- Whigham, P. A. (1995). Grammatically-based genetic programming. In Rosca, J. P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA.
- Wineberg, M. and Oppacher, F. (1996). The benefits of computing with introns. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 410–415, Stanford University, CA, USA. MIT Press.
- Wong, M. L. and Leung, K. S. (1996). Evolving recursive functions for the even-parity problem using genetic programming. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, chapter 11, pages 221–240. MIT Press, Cambridge, MA, USA.
- Zannoni, E. and Reynolds, R. G. (1997). Learning to control the program evolution process with cultural algorithms. *Evolutionary Computation*, 5(2):181–211.

An Investigation of Transformation-Based Learning in Discourse

Ken Samuel
 CIS Department
 University of Delaware
 Newark, Delaware 19716 USA
 samuel@cis.udel.edu

Sandra Carberry
 CIS Department
 University of Delaware
 Newark, Delaware 19716 USA
 carberry@cis.udel.edu

K. Vijay-Shanker
 CIS Department
 University of Delaware
 Newark, Delaware 19716 USA
 vijay@cis.udel.edu

Abstract

This paper presents results from the first attempt to apply Transformation-Based Learning to a discourse-level Natural Language Processing task. To address two limitations of the standard algorithm, we developed a Monte Carlo version of Transformation-Based Learning to make the method tractable for a wider range of problems without degradation in accuracy, and we devised a committee method for assigning confidence measures to tags produced by Transformation-Based Learning. The paper describes these advances, presents experimental evidence that Transformation-Based Learning is as effective as alternative approaches (such as Decision Trees and N-Grams) for a discourse task called Dialogue Act Tagging, and argues that Transformation-Based Learning has desirable features that make it particularly appealing for the Dialogue Act Tagging task.

1 INTRODUCTION

Transformation-Based Learning is a relatively new machine learning method, which has been as effective as any other approach on the Part-of-Speech Tagging problem¹ (Brill, 1995a). We are utilizing Transformation-Based Learning for another important language task called Dialogue Act Tagging, in which the goal is to label each utterance in a conversational dialogue with the proper dialogue act. A *dialogue act* is a concise abstraction of a speaker's intention, such as SUGGEST or ACCEPT. Recognizing dialogue acts is critical for discourse-level understanding and can also

be useful for other applications, such as resolving ambiguity in speech recognition. But computing dialogue acts is a challenging task, because often a dialogue act cannot be directly inferred from a literal reading of an utterance. Figure 1 presents a hypothetical dialogue that has been labeled with dialogue acts.

Our research efforts led us to address some limitations of Transformation-Based Learning. We developed a Monte Carlo version of the algorithm that overcomes the limitation of Transformation-Based Learning's dependence on manually-generated *rule templates* and enables Transformation-Based Learning to be applied effectively to a wider range of tasks. We also devised a technique that uses a committee of learned models to derive *confidence measures* associated with the dialogue acts assigned to utterances.

We experimentally compared our modified version of Transformation-Based Learning with C5.0, an implementation of Decision Trees, and N-Grams, which was previously the best reported method for Dialogue Act Tagging (Reithinger and Klesen, 1997). Our system performs as well as these benchmarks, and we note that Transformation-Based Learning has several characteristics that make it particularly appealing for the Dialogue Act Tagging task.

This paper begins with an overview of the Transformation-Based Learning method, describing the training phase and the application phase of the algorithm and presenting some of Transformation-Based Learning's most attractive characteristics for Dialogue Act Tagging. The following section describes the experimental design used for the experiments presented in the paper. Then Section 4 presents two limitations of Transformation-Based Learning, a dependence on rule templates and a lack of confidence measures, and describes our solutions for these problems, a Monte Carlo strategy and a committee method. Next we present an experimental comparison between Transformation-Based Learning, N-Grams, and Decision Trees, and conclude with a discussion of this work.

¹The goal of this Natural Language Processing task is to label words with the proper part of speech tags, such as Noun and Verb.

#	Speaker	Utterance	Dialogue Act
1	John	Hello.	GREET
2	John	I'd like to meet with you on Tuesday at 2:00.	SUGGEST
3	Mary	That's no good for me,	REJECT
4	Mary	but I'm free at 3:00.	SUGGEST
5	John	That sounds fine to me.	ACCEPT
6	John	I'll see you then.	BYE

Figure 1: A sample dialogue

2 TRANSFORMATION-BASED LEARNING

Brill (1995a) developed a symbolic machine learning method called Transformation-Based Learning. Given a tagged training corpus, Transformation-Based Learning produces a sequence of rules that serves as a model of the training data. Then, to derive the appropriate tags, each rule may be applied, in order, to each instance in an untagged corpus. For all of the results and examples in this paper, we are using Transformation-Based Learning on the Dialogue Act Tagging task, so the instances are utterances and the tags are dialogue acts. In one experiment, our system produced a learned model with 213 rules; the first five rules are presented in Figure 2.

#	Condition(s)	New Dialogue Act
1	<i>none</i>	SUGGEST
2	Includes "see" and "you"	BYE
3	Includes "sounds"	ACCEPT
4	Length < 4 words Previous tag is <i>none</i> ²	GREET
5	Includes "no" Previous tag is SUGGEST	REJECT

Figure 2: Rules produced by Transformation-Based Learning for Dialogue Act Tagging

2.1 THE TRAINING PHASE

The training phase of TBL, in which the system learns a sequence of rules based on a tagged training corpus, proceeds in the following manner:

1. Label each instance with a dummy tag.
2. Until no useful rules are found,
 - a. For each incorrect tag
 - i. Generate all rules that correct the tag.
 - b. Score each generated rule.
 - c. Output the highest scoring rule.
 - d. Apply this rule to the corpus.

²This condition is true only for the first utterance of a dialogue.

First, the system initializes the training corpus by labeling each instance with a dummy tag. Brill (1995a) suggested using a more complex initialization step, but we found that this simple strategy is more effective in practice.³ Then the system generates all of the *potential rules* that would make at least one tag in the training corpus correct, under the restrictions described below. For each potential rule, its *improvement score* is defined to be the number of correct tags in the training corpus after applying the rule *minus* the number of correct tags in the training corpus before applying the rule. The potential rule with the highest improvement score is output as the next rule in the final model and applied to the entire training corpus. This process repeats (using the updated tags on the training corpus), producing one rule for each pass through the training corpus until no rule can be found with an improvement score that surpasses some predefined threshold. In practice, threshold values of 1 or 2 appear to be effective.

Since there are potentially an infinite number of rules that could produce the tags in the training data, it is necessary to restrict the range of patterns that the system may consider by providing a set of rule templates, such as:

IF utterance **u** contains the word(s) **w**
 AND the tag on the utterance preceding **u** is **X**
 THEN change **u**'s tag to **Y**

This template can be instantiated to produce the last rule in Figure 2 by setting **w**="no", **X**=SUGGEST, and **Y**=REJECT.

For the first rules of the learned model, the emphasis is on getting as many tags correct as possible with no penalty imposed for changing an incorrect tag to another incorrect tag. Then for the later rules, the system must avoid changing any of the tags that are

³This is because Transformation-Based Learning uses an error-driven approach, only generating rules for the instances that are incorrectly labeled. If every instance is initialized with a dummy tag, then all of the labels are incorrect, and so they all contribute to learning. Alternatively, using a more involved initialization step results in a greater number of correct tags and, effectively, less training data.

already correct. Thus, this method tends to produce a sequence of rules that progresses from general rules to specific rules.

2.2 THE APPLICATION PHASE

To see how a rule sequence can be used to label data, consider applying the rules in Figure 2 to the dialogue in Figure 1. The first rule labels every utterance with the dialogue act SUGGEST. Next, the second rule changes an utterance's tag to BYE if it contains the words "see" and "you", which only holds for utterance #6. Similarly, the third rule changes utterance #5's tag to ACCEPT. Then the fourth rule tags utterance #1 as GREET, since its length is 1 and there is no preceding utterance in the dialogue. And finally, the last rule relabels utterance #3 as REJECT, since utterance #2 is currently tagged SUGGEST, and the word "no" is found in utterance #3. Although the first five rules label these six utterances correctly, the remaining 208 rules in the sequence may continue to adjust the tags on the utterances.

2.3 ATTRACTIVE CHARACTERISTICS

For the Dialogue Act Tagging task, we selected Transformation-Based Learning for several reasons. Brill reported that Transformation-Based Learning is as good as or better than any other algorithm for the Part-of-Speech Tagging problem, labeling 97.2% of the words correctly. The part-of-speech tag of a word is dependent on the word's internal features and on the surrounding words; similarly, the dialogue act of an utterance is dependent on the utterance's internal features and on the surrounding utterances. This parallel suggests that Transformation-Based Learning has potential for success on the Dialogue Act Tagging problem.

Since we currently lack a systematic theory of dialogue acts, another reason that Transformation-Based Learning is an attractive choice is that its learned model consists of relatively intuitive rules (Brill, 1995a), which a human can analyze to determine what the system has learned and develop a working theory. Also, Transformation-Based Learning is good at ignoring any potential rules that are irrelevant. This is because irrelevant rules tend to have a random effect on the training data, which usually results in low improvement scores, so these rules are unlikely to be selected for inclusion in the final model. This is very helpful for Dialogue Act Tagging, since we don't know what the relevant templates are for this problem. Ramshaw and Marcus (1994) experimentally demonstrated Transformation-Based Learning's robustness with respect to irrelevant rules.

For these reasons, along with others that are pre-

sented at the end of the paper, we believe that Transformation-Based Learning is worthy of investigation for the Dialogue Act Tagging task.

3 EXPERIMENTAL DESIGN

All of the results presented in this paper followed the same experimental design as the third experiment in Reithinger and Klesen (1997). The corpus consisted of appointment-scheduling face-to-face dialogues in English, which was divided into a training set with 143 dialogues (2701 utterances) and a disjoint testing set with 20 dialogues (328 utterances). Each utterance was manually labeled with one of 18 abstract dialogue acts, such as SUGGEST, ACCEPT, REJECT, GREET, and BYE. The full list of dialogue acts is found in Reithinger and Klesen (1997).

The Transformation-Based Learning experiments presented in this paper were run on a Sun Ultra 1 machine with 508MB of main memory. Within a set of experiments, only the specified parameters were varied, but between sets of experiments many parameters may have been varied, so it is not possible to draw conclusions across experiment sets.

Our rule templates consist of all possible combinations of a preselected set of conditions. Some of these conditions are presented in Figure 3. Each *condition* consists of a feature and a distance, where the *feature* specifies a characteristic of utterances that might be relevant for the Dialogue Act Tagging task, and the *distance* specifies the relative position (from the utterance under analysis) of the utterance that the feature should be applied to.

Feature	Distance	
length	of the	current utterance
tag	of the	preceding utterance
cue patterns	of the	current utterance
speaker	of the	current utterance
speaker	of the	preceding utterance

Figure 3: Some conditions used in our experiments

In discourse, it is widely acknowledged that some of the short phrases (and specific words) found in an utterance provide strong clues to determine the appropriate dialogue act. Several researchers proposed different *cue phrases*, which are phrases that appear frequently in dialogue and convey useful discourse information, such as "but", "so", and "by the way". Unfortunately, there is no universal agreement on which phrases should be considered cue phrases, and in a preliminary experiment using all of the cue phrases proposed in the literature,⁴ our system's accuracy only

⁴These lists of cue phrases can be found in Hirschberg

improved by 1.03%.

In order to identify the phrases that will be useful for a particular domain, we need an *automatic* method for collecting a set of phrases that is tuned to that domain. So we are using a statistical approach to select relevant *cue patterns*⁵ from a training corpus. Assuming that a phrase is relevant if it co-occurs frequently with a few specific dialogue acts, we analyze the distribution of dialogue acts for utterances that include a given phrase, selecting those phrases that correspond to dialogue act distributions with low entropy. When using these cue patterns, our system's accuracy rose by 17.63%. For more details on this work, see Samuel, Carberry, and Vijay-Shanker (1998b).

4 TRANSFORMATION-BASED LEARNING IN DISCOURSE

4.1 TWO LIMITATIONS

Transformation-Based Learning has two serious limitations, which we will address in this section. First, although Transformation-Based Learning produces a tag for each instance, it doesn't offer any measure of confidence in these tags. Alternatively, probabilistic machine learning approaches generally label an instance with a set of tags, which are assigned numbers to represent the likelihood that they are correct. So "probabilistic methods ... provide a continuous ranking of alternative analyses rather than just a single output, and such rankings can productively increase the bandwidth between components of a modular system." (Brill and Mooney, 1997)

The second limitation of Transformation-Based Learning is that it is highly dependent on the rule templates, which are manually developed in advance. Since the omission of any relevant templates would handicap the system, it is essential that these choices be made carefully. But in Dialogue Act Tagging, no one knows exactly which conditions and combinations of conditions are relevant, so it is preferable to err on the side of caution by constructing an overly-general set of templates and allowing the system to *learn* which templates are useful. As discussed earlier, Transformation-Based Learning is capable of discarding irrelevant rules, so this approach should be effective, in theory.

Unfortunately, this strategy is not tractable, because for each pass through the training data, for each instance that the system has tagged incorrectly, *every* rule template must be instantiated in *all* possible ways.

Suppose that we can postulate f different features that might be relevant, and we wish to consider these features for all instances that occur within a distance d of a given instance. (In other words, we are using a contextual window of size $2d+1$.) Then there are $(2d+1)f$ conditions and $2^{(2d+1)f}$ possible templates, since each condition may either be included or excluded. Also, suppose that when a feature is applied to an instance, it produces v distinct values, on average. This results in $(v+1)^{(2d+1)f}$ rules per instance, which can be proven by induction on the number of conditions. Given a training corpus with i instances, if the algorithm makes p passes through the training data, then the system must generate and evaluate $O(ip(v+1)^{(2d+1)f})$ rules. Some realistic values for these variables are $f=10$, $d=2$ (a contextual window of size 5), $v=3$, $i=3000$, and $p=100$, which generates around 10^{35} rules. Based on experimental evidence, it appears that it is necessary to drastically limit the number of potential rules that the system generates,⁶ or the memory and time costs are so exorbitant that the method becomes intractable. But this limitation would preclude considering all of the features and feature interactions that might be relevant for Dialogue Act Tagging.

4.2 A MONTE CARLO VERSION

We developed a Monte Carlo version of Transformation-Based Learning, so that the system can consider a huge number of templates while still maintaining tractability. Rather than exhaustively searching through the space of possible rules, only R of the available template instantiations are randomly selected for each training instance on each pass through the training data, where R is some small integer. With this modification, the total number of rules generated is only $O(ipR)$, which no longer explodes with the number of templates. In fact, the formula doesn't even depend on the number of features, the contextual window size, or the value of v . But one would still expect good results, because Transformation-Based Learning only needs to find the best rules, and the best rules tend to be effective for a large number of different instances. So the system has many opportunities to find these rules, and since the algorithm generally makes many passes through the training data before halting, if it should select a suboptimal rule, it can use later rules to compensate. Thus, although random sampling will miss some rules, it is still highly likely to find an effective sequence of rules.

Our experiments confirm these intuitions, as shown in Figures 4 and 5. For these runs, eight condi-

and Litman (1993) and Knott (1996).

⁵In practice, the concept of cue patterns tends to be more general than cue phrases, including many more phrases.

⁶For the Part-of-Speech Tagging task, Brill used only about 30 simple rule templates (Brill, 1995a).

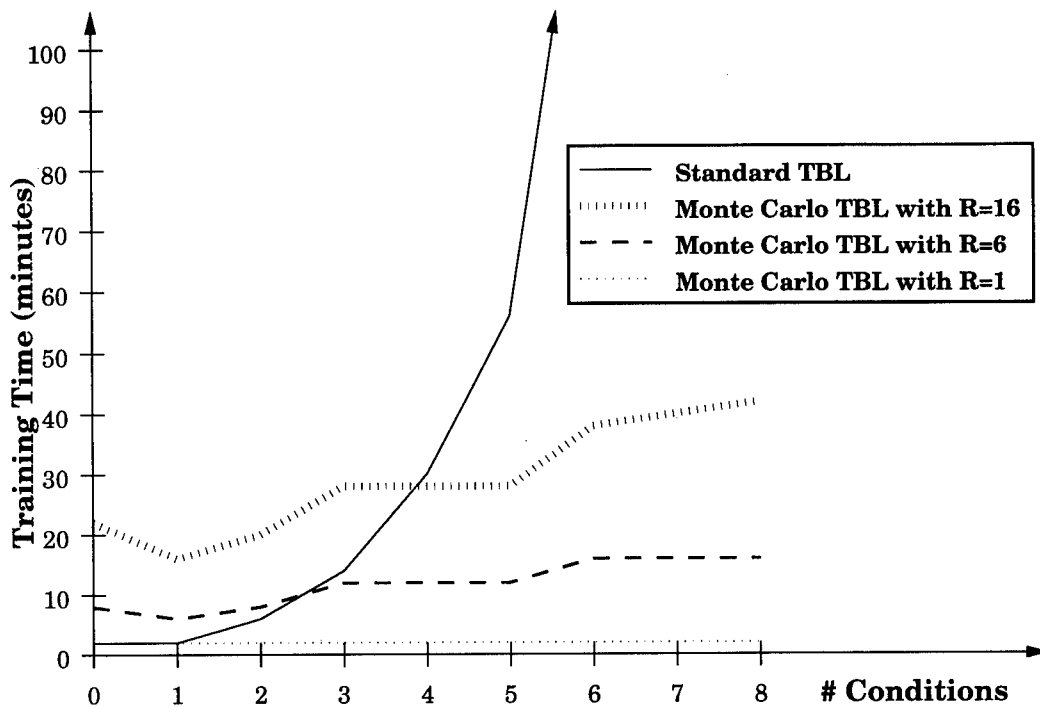


Figure 4: Number of conditions vs. training time

tions were preselected, and for different values of n , $0 \leq n \leq 8$, the first n conditions were combined in all possible ways to generate 2^n templates. Using these templates, we trained, tested, and compared the standard Transformation-Based Learning method and our Monte Carlo version of Transformation-Based Learning.

For the standard Transformation-Based Learning method, training time rises dramatically as the number of conditions increases, as shown in Figure 4.⁷ In fact, when given seven conditions, the standard Transformation-Based Learning algorithm could not complete the training phase, even after running for more than 24 hours. But our Monte Carlo version of Transformation-Based Learning keeps the efficiency relatively stable.⁸ The reason for the slight increase in training time as the number of conditions increases is

⁷The value of v (the average number of rules generated per instance) varies slightly across the eight conditions, and so the shape of the curve might vary depending on the order in which the conditions are presented. But the critical point is that the training time rises exponentially with the number of conditions.

⁸The Monte Carlo version of Transformation-Based Learning can be slower than the standard method, because the Monte Carlo version always generates R rules for each instance, without checking for repetitions. (It would be too inefficient to prevent the system from generating any rule more than once.)

that, as the system gains access to a greater number of useful conditions, it's likely to find a greater number of useful rules, meaning that the training phase makes a greater number of passes through the training data. Thus, p increases, and so the training time, $O(ipR)$, also increases. But this increase is linear (or less), while standard Transformation-Based Learning's training time increases exponentially with the number of conditions. Figure 4 supports this analysis.

This improvement in time efficiency would be quite uninteresting if the performance of the algorithm deteriorated significantly. But, as Figure 5 shows, this is not the case. Although setting R too low (such as $R=1$ for 7 and 8 conditions) may result in a decrease in accuracy, the lowest possible setting ($R=1$) is as accurate as standard Transformation-Based Learning for 6 conditions (64 templates). For 7 and 8 conditions, training of the standard Transformation-Based Learning method took too much time, so those results could not be produced. But, as the curves for $R=6$ and $R=16$ do not differ significantly, it is reasonable to predict that standard Transformation-Based Learning would produce similar results as well.⁹ Therefore, we conclude

⁹One might wonder how the Monte Carlo version of Transformation-Based Learning can ever do better than the standard Transformation-Based Learning method, which occurred for the experiments that used five conditions. Because Transformation-Based Learning is a greedy algorithm, choosing the best available rule on each

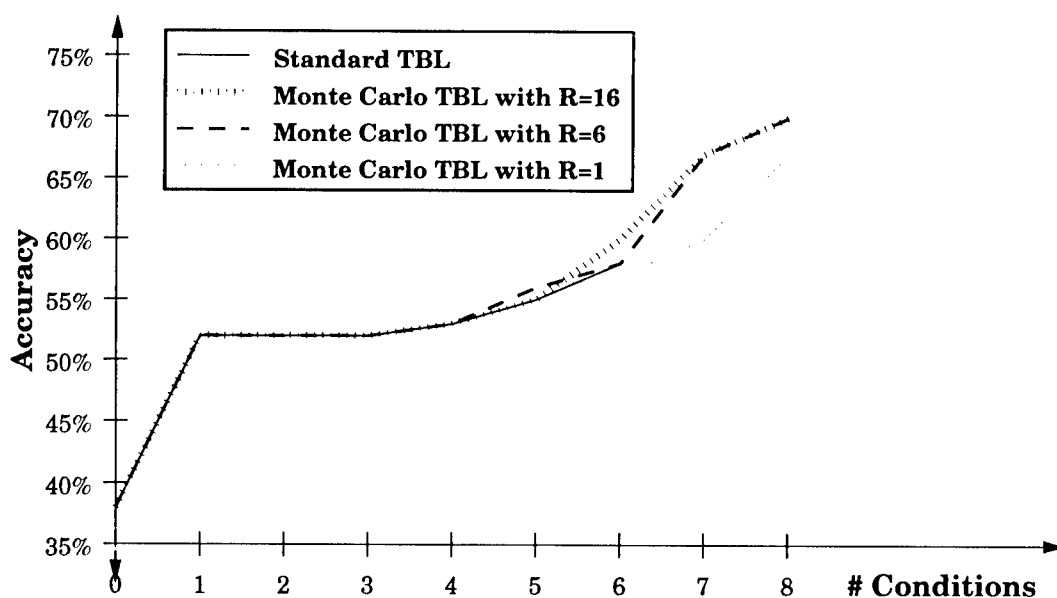


Figure 5: Number of conditions vs. tagging accuracy on unseen data

that our Monte Carlo version of Transformation-Based Learning (with $R=6$) works effectively for more than 250 templates (8 conditions) in only about 15 minutes of training time.

4.3 A COMMITTEE METHOD

We wanted to extend Transformation-Based Learning so that it could provide some idea of the likelihood that each of its tags are correct. So we attempted to develop a strategy for assigning confidence measures to the *rules* in the learned model. Then, in the application phase, a given instance's confidence measure would be a function of the confidences of the rules that applied to that instance. Unfortunately, due to the nature of the Transformation-Based Learning method, this straightforward approach has been unsuccessful, because the rule sequence does not contain enough information to derive confidence measures; often, the same pattern of rules applies to instances that should be marked with high confidence as well as instances that should be marked with low confidence.

So, for the purpose of computing confidence measures, we adapted two techniques that were developed for very different tasks. The Boosting approach has been used to improve accuracy in tagging data (Freund and Schapire, 1996), and Committee-Based Sampling utilized a very similar strategy to minimize the required

pass through the training data, sometimes the standard Transformation-Based Learning method selects a rule that locks it into a local maximum, while the Monte Carlo version might fail to consider this attractive rule and end up producing a better model.

size of a training corpus (Dagan and Engelson, 1995). We applied these methods to compute confidence measures, by training the system a number of times to produce a few different but reasonable learned models, which are called *committee members*. Then given new data, each committee member independently tags the input, and a given tag's confidence is based on how well the committee members agree on that tag. We are currently defining the confidence of a given tag to be the number of committee members that preferred the tag. In the future, we will investigate confidence formulas that are based on the entropy of the tags selected by the different committee members.

We considered several ways to develop the committee members, and we decided to apply the strategy that Freund and Schapire (1996) used for Boosting: The first committee member is trained in the standard way, and then the second committee member pays special attention to those instances in the training data that the first committee member did not tag correctly. To do this in Transformation-Based Learning, we adjust the improvement score formula to weight success on these "hard" instances more heavily. (In effect, it is as if we were adding multiple copies of these instances to the training corpus.) This process can be repeated to generate more committee members by basing the score for correctly tagging a training instance on the number of previous committee members that tagged that instance incorrectly. We are currently using 2^c as the score for correctly tagging a given instance that c committee members have mistagged. This strategy tends to produce committee members that are very different, as they are focusing on different parts of the

training corpus.

Minimum Confidence	Percentage of Instances Tagged	Average Precision
5	45.12% \pm 1.28%	90.09% \pm 1.51%
4	69.79% \pm 1.60%	83.53% \pm 1.27%
3	92.38% \pm 1.32%	76.57% \pm 0.79%
2	99.85% \pm 0.20%	73.56% \pm 1.10%
1	100.00% \pm 0.00%	73.45% \pm 1.06%

Figure 6: Testing the committee method on unseen data, varying the minimum confidence considered

As a preliminary experiment we ran ten trials with five committee members, testing on held-out data. Figure 6 presents average scores and standard deviations, varying the minimum confidence, *m*. For a given instance, if at least *m* committee members agreed on a tag, then the most popular tag was applied, breaking ties in favor of the committee member that was developed the earliest; otherwise no tag was output. The results show that the committee approach assigns useful confidence measures to the tags: All five committee members agreed on the tags for 45.12% of the instances, and 90.09% of those tags were correct. Also, for 69.79% of the instances, at least four of the five committee members selected the same tag, and this tag was correct 83.53% of the time. We foresee that our module for tagging dialogue acts can potentially be integrated into a larger system so that, when Transformation-Based Learning cannot produce a tag with high confidence, other modules may be invoked to provide more evidence. In addition, like Boosting, the committee method improves the overall accuracy of the system. By selecting the most popular tag among all five committee members, the average accuracy in tagging unseen data was 73.45%, while using the first committee member alone resulted in a significantly ($t = 5.42 > 2.88$, $\alpha = 0.01$) lower average score of 70.79%.

4.4 ALTERNATIVE METHODS

Previously, the best success rate achieved on the Dialogue Act Tagging problem was reported by Reithinger and Klesen (1997), whose system used a probabilistic machine learning approach based on N-Grams to correctly label 74.7% of the utterances in a test corpus. (See Samuel, Carberry, and Vijay-Shanker (1998a) for a more extensive analysis of previous work on this task.) As a direct comparison, we applied our system to exactly the same training and testing set. Over five runs, the system achieved an average¹⁰ accuracy of 75.12% \pm 1.34%, including a high score¹¹ of 77.44%.

¹⁰The variation in the scores is due to the random nature of the Monte Carlo method.

¹¹The rules in Figure 2 were produced in this experiment.

In addition, we ran a direct comparison between Transformation-Based Learning and C5.0 (Rulequest Research, 1998), which is an implementation of the Decision Trees method. The accuracies on held-out data for training sets of various sizes are presented in Figure 7. For Transformation-Based Learning, we averaged the scores of ten trials for each training set (to factor out the random effects of the Monte Carlo method), and the standard deviations are represented by error bars in the graph. These experiments did not utilize the committee method, and we would expect the scores to improve when this extension is used.

With C5.0, we wanted to use the same features that were effective for Transformation-Based Learning, but we encountered two problems: 1) Since C5.0 requires that each feature take exactly one value for each instance, it is very difficult to utilize the cue patterns feature. We decided to provide one boolean feature for each possible cue pattern, which was set to True for instances that included that cue pattern and False otherwise. 2) Our Transformation-Based Learning system utilized the system-generated tag¹² of the preceding instance. C5.0 cannot use this information, as it requires that the values of all of the features are computed before training begins.

The training times of Transformation-Based Learning and C5.0 were relatively comparable for any number of conditions, although Boosting sometimes resulted in a significant increase in training time. The accuracy scores of Transformation-Based Learning and C5.0, with and without Boosting, are not significantly different, as shown in Figure 7.

5 DISCUSSION

This paper has described the first investigation of Transformation-Based Learning applied to discourse-level problems. We extended the algorithm to address two limitations of Transformation-Based Learning: 1) We developed a Monte Carlo version of Transformation-Based Learning, and our experiments suggest that this improvement dramatically increases the efficiency of the method without compromising accuracy. This revision enables Transformation-Based Learning to work effectively on a wider variety of tasks, including tasks where the relevant conditions and condition combinations are not known in advance as well as tasks where there are a large number of relevant conditions and condition combinations. This improvement also decreases the labor demands on the human developer, who no longer needs to construct a mini-

¹²For Transformation-Based Learning, the tags change as the system applies the rules in the learned model. When a rule references a tag, it uses the value of the tag at the point when that rule is processed.

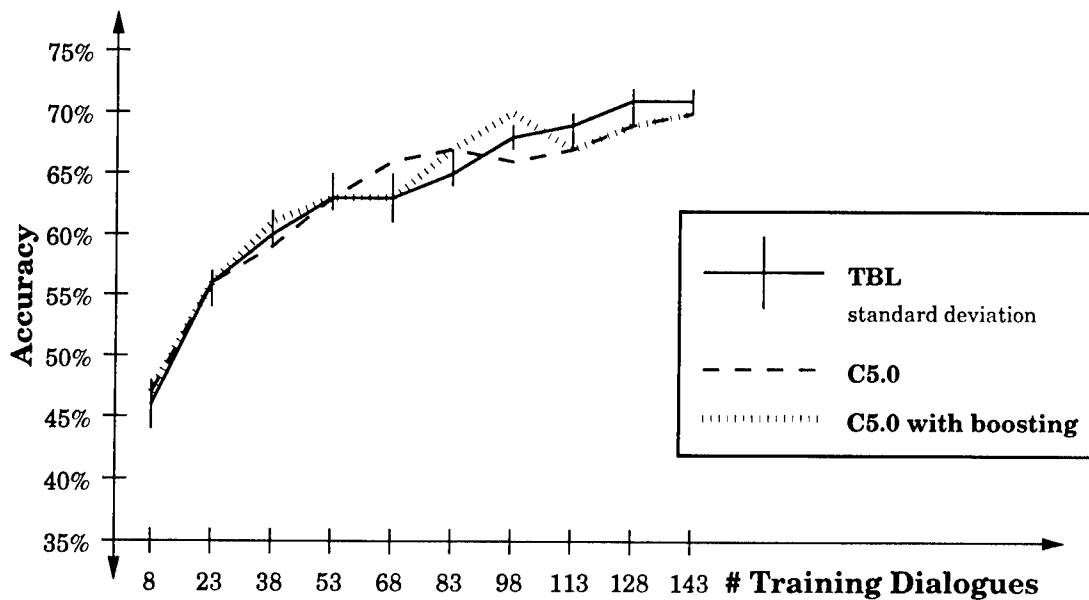


Figure 7: Training set size vs. tagging accuracy on unseen data

mal set of rule templates. It is sufficient to list all of the conditions that might be relevant and allow the system to consider all possible combinations of those conditions. 2) We devised a committee strategy for computing confidence measures to represent the reliability of tags. In our experiments, this committee method improved the overall tagging accuracy significantly. It also produced useful confidence measures; nearly half of the tags were assigned high confidence, and of these, 90% were correct.

For the Dialogue Act Tagging task, our modified version of Transformation-Based Learning has achieved an accuracy rate that is comparable to any previously reported system. In addition, Transformation-Based Learning has a number of features that make it particularly appealing for the Dialogue Act Tagging task:

1. Transformation-Based Learning's learned model consists of a relatively short sequence of intuitive rules, stressing relevant features and highlighting important relationships between features and tags (Brill, 1995a). Thus, Transformation-Based Learning's learned model offers insights into a *theory* to explain the training data. This is especially useful in Dialogue Act Tagging, which currently lacks a systematic theory.
2. With its iterative training algorithm, when developing a new rule, Transformation-Based Learning can consider tags that have been produced by previous rules (Ramshaw and Marcus, 1994). Since the dialogue act of an utterance is affected by the surrounding dialogue acts, this leveraged learning approach can directly integrate the relevant

contextual information into the rules. In addition, Transformation-Based Learning can accommodate the focus shifts that frequently occur in discourse by utilizing features that consider tags of varying distances.

3. Our Transformation-Based Learning system is very flexible with respect to the types of features it can utilize. For example, it can learn set-valued features, such as cue patterns. Additionally, because of the Monte Carlo improvement, our system can handle a very large number of features.
4. For the Dialogue Act Tagging task, people still don't know what features are relevant, so it is very difficult to construct an appropriate set of rule templates. Fortunately, Transformation-Based Learning is capable of discarding irrelevant rules, as Ramshaw and Marcus (1994) showed experimentally, so it is not necessary that *all* of the given rule templates be useful.
5. Ramshaw and Marcus's (1994) experiments suggest that Transformation-Based Learning tends to be resistant to the overfitting¹³ problem. This can be explained by observing how the rule sequence produced by Transformation-Based Learning progresses from general rules to specific rules. The early rules in the sequence are based on many examples in the training corpus, and so they are likely to generalize effectively to new data. Later in the sequence, the rules don't receive as much

¹³Other machine learning algorithms may overfit to the training data and then have difficulty generalizing to new data.

support from the training data, and their applicability conditions tend to be very specific, so they have little or no effect on new data. Thus, resistance to overfitting is an emergent property of the Transformation-Based Learning algorithm.

For the future, we intend to investigate a wider variety of features and explore different methods for collecting cue patterns to increase our system's accuracy scores further. Although we compared Transformation-Based Learning with a few very different machine learning algorithms, we still hope to examine other methods, such as Naive Bayes. In addition, we plan to run our experiments with different corpora to confirm that the encouraging results of our extensions to Transformation-Based Learning can be generalized to different data, languages, domains, and tasks. We would also like to extend our system so that it may learn from untagged data, as there is still very little tagged data available in discourse. Brill developed an unsupervised version of Transformation-Based Learning for Part-of-Speech Tagging (Brill, 1995b), but this algorithm must be initialized with instances that can be tagged unambiguously (such as "the", which is always a determiner), and in Dialogue Act Tagging there are very few unambiguous examples. We intend to investigate the following weakly-supervised approach: First, the system will be trained on a small set of tagged data to produce a number of different committee members. Then given untagged data, it will derive tags with confidence measures. Those tags that receive very high confidence can be used as unambiguous examples to drive the unsupervised version of Transformation-Based Learning.

Acknowledgments

We wish to thank the members of the VERBMOBIL research group at DFKI in Germany, particularly Norbert Reithinger, Jan Alexandersson, and Elisabeth Maier, for providing the first author with the opportunity to work with them and generously granting him access to the VERBMOBIL corpora. This work was partially supported by the NSF Grant #GER-9354869.

References

- Brill, Eric (1995a). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics* 21(4):543-566.
- Brill, Eric (1995b). Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In *Proceedings of the Very Large Corpora Workshop*.
- Brill, Eric and Mooney, Raymond J. (1997). An Overview of Empirical Natural Language Processing. *AI Magazine* 18(4):13-24.
- Dagan, Ido and Engelson, Sean P. (1995). Committee-Based Sampling for Training Probabilistic Classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*.
- Freund, Yoav and Schapire, Robert E. (1996). Experiments with a New Boosting Algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*.
- Hirschberg, Julia and Litman, Diane (1993). Empirical Studies on the Disambiguation of Cue Phrases. *Computational Linguistics* 19(3):501-530.
- Knott, Alistair (1996). A Data-Driven Methodology for Motivating a Set of Coherence Relations. *Ph.D. Thesis*. The University of Edinburgh.
- Ramshaw, Lance A. and Marcus, Mitchell P. (1994). Exploring the Statistical Derivation of Transformation Rule Sequences for Part-of-Speech Tagging. In *Proceedings of the 32nd Annual Meeting of the ACL*.
- Reithinger, Norbert and Klesen, Martin (1997). Dialogue Act Classification Using Language Models. In *Proceedings of EuroSpeech-97*.
- Rulequest Research. (1998). Data Mining Tools see5 and c5.0. [<http://www.rulequest.com/see5-info.html>].
- Samuel, Ken, Carberry, Sandra, and Vijay-Shanker, K. (1998a). Computing Dialogue Acts from Features with Transformation-Based Learning. In *Applying Machine Learning to Discourse Processing: Papers from the 1998 AAAI Spring Symposium*.
- Samuel, Ken, Carberry, Sandra, and Vijay-Shanker, K. (1998b). Dialogue Act Tagging with Transformation-Based Learning. In *Proceedings of COLING-ACL*.

Automatic Segmentation of Continuous Trajectories with Invariance to Nonlinear Warpings of Time

Lawrence K. Saul
AT&T Labs — Research
180 Park Ave, E-171
Florham Park, NJ 07932
lsaul@research.att.com

Abstract

We study the classification problem that arises when two variables—one continuous (\mathbf{x}), one discrete (s)—evolve jointly in time. We suppose that the vector \mathbf{x} traces out a smooth multidimensional curve, to each point of which the variable s attaches a discrete label. The trace of s thus partitions the curve into different *segments* whose boundaries occur where s changes value. We consider how to learn the mapping between \mathbf{x} and s from examples of segmented curves. Our approach is to model the conditional random process that generates segments of constant s along the curve of \mathbf{x} . We suppose that the variable s evolves stochastically as a function of the *arc length* traversed by \mathbf{x} . Since arc length does not depend on the rate at which a curve is traversed, this gives rise to a family of Markov processes whose predictions, $\Pr[s|\mathbf{x}]$, are invariant to nonlinear warpings (or reparameterizations) of time. We show how to learn the parameters of these Markov processes from labeled and/or unlabeled examples of segmented curves. The resulting models are motivated for automatic speech recognition, where \mathbf{x} are acoustic features and s are phonetic transcriptions.

1 INTRODUCTION

The automatic segmentation of continuous trajectories poses a challenging problem in machine learning. The problem arises whenever a multidimensional trajectory $\{\mathbf{x}(t)|t \in [0, \tau]\}$ must be described by a se-

quence of discrete labels $s_1 s_2 \dots s_n$. A simple way to map trajectories into sequences is to specify consecutive time intervals such that $s(t) = s_k$ for $t \in [t_{k-1}, t_k]$. This attaches the labels s_k to contiguous arcs along the trajectory. The learning problem is to discover such a mapping from labeled and/or unlabeled examples.

In this paper, we study this problem, paying special attention to the fact that curves have intrinsic geometric properties that do not depend on the rate at which they are traversed (do Carmo, 1976). Such properties include, for example, the total arc length and the maximum distance between any two points on the curve. Given a multidimensional trajectory $\{\mathbf{x}(t)|t \in [0, \tau]\}$, these properties are invariant to reparameterizations $t \rightarrow f(t)$, where $f(t)$ is any monotonic function that maps the interval $[0, \tau]$ into itself. Put another way, the intrinsic geometric properties of the curve are invariant to *nonlinear warpings of time*.

Invariance to nonlinear warpings of time is an example of a mathematical symmetry. The importance of such symmetries in statistical pattern recognition (Duda & Hart, 1973) is well-known. For example, in the problem of object recognition from two dimensional images, one often incorporates invariances to translations, rotations, and changes of scale (Simard *et al*, 1993). In the segmentation of continuous trajectories, one naturally encounters the question of invariance to nonlinear warpings of time. A better understanding of this invariance is therefore valuable in its own right. Beyond its mathematical interest, however, the principled handling of this invariance suggests new algorithms for the automatic segmentation of continuous trajectories. Indeed, the primary motivation for this work is its potential application to automatic speech recognition—a subject to which we return in the final section of the paper.

The study of curves requires some simple notions from

differential geometry. As a matter of terminology, we refer to particular parameterizations of curves as trajectories. We regard two trajectories $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ as equivalent to the same curve if there exists a monotonically increasing function f for which $\mathbf{x}_1(t) = \mathbf{x}_2(f(t))$. (To be precise, we mean the same *oriented* curve: the direction of traversal matters.) Here, as in what follows, we adopt the convention of using $\mathbf{x}(t)$ to denote an entire trajectory as opposed to constantly writing out $\{\mathbf{x}(t) | t \in [0, \tau]\}$. When necessary to refer to the value of $\mathbf{x}(t)$ as a particular moment in time, we use a different index, such as $\mathbf{x}(t_1)$.

Let us return now to the problem of automatic segmentation. Consider two variables—one continuous (\mathbf{x}), one discrete (s)—that evolve jointly in time. Thus the vector \mathbf{x} traces out a smooth multidimensional curve, to each point of which the variable s attaches a discrete label. Note that each trace of s yields a partition of the curve into different components; in particular, the boundaries of these components occur at the points where s changes value. We refer to such partitions as segmentations and to the regions of constant s as *segments*; see figure 1.

Our goal in this paper is to learn a probabilistic mapping between trajectories $\mathbf{x}(t)$ and segmentations $s(t)$ from labeled and/or unlabeled examples. Consider the conditional random process that generates segments of constant s along the curve traced out by \mathbf{x} . Given a trajectory $\mathbf{x}(t)$, let $\Pr[s(t) | \mathbf{x}(t)]$ denote the conditional probability distribution over possible segmentations. Suppose that for any two equivalent trajectories $\mathbf{x}(t)$ and $\mathbf{x}(f(t))$, we have the identity:

$$\Pr[s(t) | \mathbf{x}(t)] = \Pr[s(f(t)) | \mathbf{x}(f(t))]. \quad (1)$$

Eq. (1) captures a fundamental invariance—namely, that the probability that the curve is segmented in a particular way is independent of the rate at which it is traversed. In this paper, we study Markov processes with this property. We call them *Markov processes on curves* (MPCs) because for these processes it is unambiguous to write $\Pr[s | \mathbf{x}]$ without providing explicit parameterizations for the trajectories, $\mathbf{x}(t)$ or $s(t)$. The distinguishing feature of MPCs is that the variable s evolves as a function of the *arc length* traversed along \mathbf{x} , a quantity that is manifestly invariant to nonlinear warpings of time.

The main contributions of this paper are: (i) to postulate eq. (1) as a fundamental invariance of random processes; (ii) to introduce MPCs as a family of probabilistic models that capture this invariance; (iii) to derive monotonically convergent learning procedures

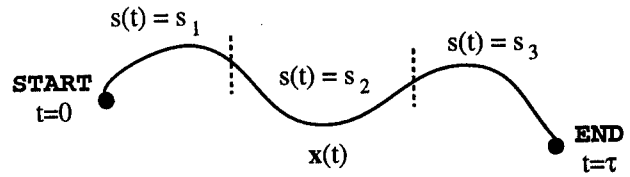


Figure 1: Two variables—one continuous (\mathbf{x}), one discrete (s)—evolve jointly in time. The trace of s partitions the curve of \mathbf{x} into different segments whose boundaries occur where s changes value. Markov processes on curves model the conditional distribution, $\Pr[s | \mathbf{x}]$.

for MPCs based on the principle of maximum likelihood estimation; and (iv) to contrast the properties of MPCs with those of hidden Markov models (HMMs), especially as they relate to problems in automatic speech recognition (Rabiner & Juang, 1993). In terms of previous work, our motivation most closely resembles that of Tishby (1990), who several years ago proposed a dynamical system approach to speech processing.

The organization of this paper is as follows. In section 2, we begin by reviewing some basic concepts from differential geometry. We then introduce MPCs as a family of continuous-time Markov processes that parameterize the conditional probability distribution, $\Pr[s | \mathbf{x}]$. The processes are derived from a set of differential equations that describe the pointwise evolution of s along the curve traced out by \mathbf{x} .

In section 3, we consider how to learn the parameters of MPCs in both supervised and unsupervised settings. These settings correspond to whether the learner has access to labeled or unlabeled examples. Labeled examples consist of trajectories $\mathbf{x}(t)$, along with their corresponding segmentations:

$$\{\text{START} \rightarrow (s_1, t_1) \cdots (s_n, t_n) \rightarrow \text{END}\}. \quad (2)$$

The ordered pairs in eq. (2) indicate that $s(t)$ takes the value s_k between times t_{k-1} and t_k ; the START and END states are used to mark endpoints. Unlabeled examples consist only of the trajectories $\mathbf{x}(t)$ and the boundary values:

$$\{(0, \text{START}) \rightarrow (\tau, \text{END})\}. \quad (3)$$

Eq. (3) specifies only that the Markov process starts at time $t=0$ and terminates at some later time τ . In this case, the learner must infer its own target values for $s(t)$ in order to update its parameter estimates. We view both types of learning as instances of maximum

likelihood estimation and describe an Expectation-Maximization (EM) algorithm for the more general case of unlabeled (or partially labeled) examples.

In section 4, we discuss the application of MPCs to automatic speech recognition (Rabiner & Juang, 1993). Here we can identify the curves \mathbf{x} with time-varying spectral signatures and the segmentations s with phonetic transcriptions. We discuss possible advantages of MPCs over hidden Markov models, the current leading technology for automatic speech recognition. The most important of these are: (i) the natural handling of variations in speaking rate—i.e., the rate at which acoustic features (summarized by \mathbf{x}) change with time—and (ii) the emphasis on learning a *recognition* model $\Pr[s|\mathbf{x}]$, as opposed to a *synthesis* model $\Pr[\mathbf{x}|s]$. Finally, we conclude by outlining our plans for future work.

2 MARKOV PROCESSES ON CURVES

Markov processes on curves are based fundamentally on the notion of *arc length*. After reviewing how to compute arc lengths along curves, we show how they can be used to define random processes that capture the invariance of eq. (1).

2.1 ARC LENGTH

Let $g(\mathbf{x})$ define a $D \times D$ matrix-valued function over $\mathbf{x} \in \mathcal{R}^D$. If $g(\mathbf{x})$ is everywhere non-negative definite, then we can use it as a *metric* to compute distances along curves. In particular, consider two nearby points separated by the infinitesimal vector $d\mathbf{x}$. We define the squared distance between these two points as:

$$d\ell^2 = d\mathbf{x}^T g(\mathbf{x}) d\mathbf{x}. \quad (4)$$

Arc length along a curve is the non-decreasing function computed by integrating these local distances. Thus, for the trajectory $\mathbf{x}(t)$, the arc length between the points $\mathbf{x}(t_1)$ and $\mathbf{x}(t_2)$ is given by:

$$\ell = \int_{t_1}^{t_2} dt \left[\dot{\mathbf{x}}^T g(\mathbf{x}) \dot{\mathbf{x}} \right]^{\frac{1}{2}}, \quad (5)$$

where $\dot{\mathbf{x}} = \frac{d}{dt}[\mathbf{x}(t)]$ denotes the time derivative of \mathbf{x} . Note that the arc length between two points is invariant under reparameterizations of the trajectory, $\mathbf{x}(t) \rightarrow \mathbf{x}(f(t))$, where $f(t)$ is any smooth monotonic function of time that maps the interval $[t_1, t_2]$ into itself.

In the special case where $g(\mathbf{x})$ is the identity matrix, eq. (5) reduces to the standard definition of arc length in Euclidean space. More generally, however, eq. (4) defines a non-Euclidean metric for computing arc lengths. Thus, for example, if the metric $g(\mathbf{x})$ varies as a function of \mathbf{x} , then eq. (5) can assign different arc lengths to the trajectories $\mathbf{x}(t)$ and $\mathbf{x}(t) + \mathbf{x}_0$, where \mathbf{x}_0 is a constant displacement.

2.2 STATES AND LIFETIMES

The problem of segmentation is to map a trajectory $\mathbf{x}(t)$ into a sequence of discrete labels $s_1 s_2 \dots s_n$. If these labels are attached to contiguous arcs along the curve of \mathbf{x} , then we can describe this sequence by a piecewise constant function of time, $s(t)$, as in figure 1. We refer to the possible values of s as *states*. In what follows, we introduce a family of conditional random processes that evolve s as a function of the arc length traversed along the curve traced out by \mathbf{x} . These random processes are based on a simple premise—namely, that *the probability of remaining in a particular state decays exponentially with the cumulative arc length traversed in that state*. The signature of a state is the particular way in which it computes arc length.

To formalize this idea, we associate with each state i the following quantities: (i) a position-dependent matrix $g_i(\mathbf{x})$ that can be used to compute arc lengths, as in eq. (5); (ii) a decay parameter λ_i that measures the probability per unit arc length that s makes a transition from state i to some other state; and (iii) a set of transition probabilities a_{ij} , where a_{ij} represents the probability that—having decayed out of state i —the variable s makes a transition to state j . Thus, a_{ij} defines a stochastic transition matrix with zero elements along the diagonal and rows that sum to one: $a_{ii} = 0$ and $\sum_j a_{ij} = 1$.

Together, these quantities can be used to define a Markov process along the curve traced out by \mathbf{x} . In particular, let $p_i(t)$ denote the probability that s is in state i at time t , based on its history up to that point in time. A Markov process is defined by the set of differential equations:

$$\frac{dp_i}{dt} = -\lambda_i p_i \left[\dot{\mathbf{x}}^T g_i(\mathbf{x}) \dot{\mathbf{x}} \right]^{\frac{1}{2}} + \sum_{j \neq i} \lambda_j p_j a_{ji} \left[\dot{\mathbf{x}}^T g_j(\mathbf{x}) \dot{\mathbf{x}} \right]^{\frac{1}{2}}. \quad (6)$$

The right hand side of eq. (6) consists of two competing terms. The first term computes the probability that s decays out of state i ; the second computes the probability that s decays into state i . Both probabilities are proportional to measures of arc length, and

combining them gives the overall change in probability that occurs in the time interval $[t, t + dt]$. The process is Markovian because the evolution of p_i depends only on quantities available at time t ; thus the future is independent of the past given the present.

Eq. (6) has certain properties of interest. First, note that summing both sides over i gives the identity $\sum_i dp_i/dt = 0$. This shows that p_i remains a normalized probability distribution: i.e., $\sum_i p_i = 1$ at all times. Second, suppose that we start in state i and do not allow return visits: i.e., $p_i = 1$ and $a_{ji} = 0$ for all j . In this case, the second term of eq. (6) vanishes, and we obtain a simple, one-dimensional linear differential equation for $p_i(t)$. It follows that the probability of remaining in state i decays exponentially with the amount of arc length traversed by \mathbf{x} , where arc length is computing using the matrix $g_i(\mathbf{x})$. The decay parameter, λ_i , controls the typical amount of arc length traversed in state i ; it may be viewed as an inverse lifetime or—to be more precise—an inverse *lifelength*. Finally, noting that arc length is a reparameterization-invariant quantity, we therefore observe that these dynamics capture the fundamental invariance of eq. (1).

2.3 INFERENCE

Let a_{0i} denote the probability that the variable s makes an immediate transition from the START state—denoted by the zero index—to state i ; put another way, this is the probability that the first segment belongs to state i . Given a trajectory $\mathbf{x}(t)$, the Markov process in eq. (6) gives rise to a conditional probability distribution over possible segmentations, $s(t)$. Consider the segmentation in which $s(t)$ takes the value s_k between times t_{k-1} and t_k , and let

$$\ell_{s_k} = \int_{t_{k-1}}^{t_k} dt \left[\dot{\mathbf{x}}^T g_{s_k}(\mathbf{x}) \dot{\mathbf{x}} \right]^{\frac{1}{2}} \quad (7)$$

denote the arc length traversed in state s_k . From eq. (6), we know that the probability of remaining in a particular state decays exponentially with this arc length. Thus, the conditional probability of this segmentation is given by:

$$\Pr[s|\mathbf{x}] = \prod_{k=1}^n \lambda_{s_k} e^{-\lambda_{s_k} \ell_{s_k}} \prod_{k=0}^n a_{s_k s_{k+1}}, \quad (8)$$

where we have used s_0 and s_{n+1} to denote the START and END states of the Markov process. The first product in eq. (8) multiplies the probabilities that each segment traverses exactly its observed arc length. The

second product multiplies the probabilities for transitions between states s_k and s_{k+1} . The leading factors of λ_{s_k} are included to normalize each state's duration model.

There are many important quantities that can be computed from the distribution, $\Pr[s|\mathbf{x}]$. Of particular interest is the most probable segmentation:

$$s^* = \arg \max_s \left\{ \ln \Pr[s|\mathbf{x}] \right\}. \quad (9)$$

Given a particular trajectory $\mathbf{x}(t)$, eq. (9) calls for a maximization over all piecewise constant functions of time, $s(t)$. In practice, this maximization can be performed by discretizing the time axis and applying a dynamic programming procedure. The resulting segmentations will be optimal at some finite temporal resolution, Δt . For example, let $\alpha_i(t)$ denote the log-likelihood of the most probable segmentation, ending in state i , of the subtrajectory up to time t . Starting from the initial condition $\alpha_i(0) = \ln[a_{0i}]$, we compute

$$\alpha_j(t + \Delta t) = \max_i \left\{ \alpha_i(t) - \lambda_i \Delta t \left[\dot{\mathbf{x}}^T g_i(\mathbf{x}) \dot{\mathbf{x}} \right]^{\frac{1}{2}} + \ln[\lambda_i a_{ij}](1 - \delta_{ij}) \right\}, \quad (10)$$

where δ_{ij} is the discrete delta function. Also, at each time step, let $\Psi_j(t + \Delta t)$ record the value of i that maximizes the right hand side of eq. (10). Suppose that the Markov process terminates at time τ . Enforcing the endpoint condition $s^*(\tau) = \text{END}$, we find the most likely segmentation by back-tracking:

$$s^*(t - \Delta t) = \Psi_{s^*(t)}(t). \quad (11)$$

These recursions yield a segmentation that is optimal at some finite temporal resolution Δt . Generally speaking, by choosing Δt to be sufficiently small, one can minimize the errors introduced by discretization. In practice, one would choose Δt to reflect the time scale beyond which it is not necessary to consider changes of state.

Other types of inferences can also be made from the distribution, eq. (8). For example, one can compute the marginal probability that the Markov process terminates at precisely the observed time. This is done by summing the probabilities

$$\Pr[s(\tau) = \text{END} | \mathbf{x}(t)] = \sum_{s(t)} \Pr[s(t) | \mathbf{x}(t)] \times \begin{cases} 1 & \text{if } s(\tau) = \text{END} \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

where the zero-one weighting factor selects out only those segmentations that terminate precisely at time τ . Similarly, one can compute the posterior probability, $\Pr[s(t_1) = i | \mathbf{x}(t), s(\tau) = \text{END}]$, that at an earlier moment in time, t_1 , the variable s was in state i . Both types of inferences are handled by discretizing the time axis and applying a dynamic programming procedure similar to eqs. (10–11). In the interest of brevity, we do not give the details of these constructions, noting only that in most respects they are completely analogous to the ones for discrete-time hidden Markov models (Rabiner & Juang, 1993).

3 LEARNING FROM EXAMPLES

In this section, we consider how to learn Markov processes of the form, eq. (6). By learning, we mean how to estimate the parameters $\{\lambda_i, a_{ij}, g_i(\mathbf{x})\}$ from examples of segmented (or non-segmented) curves. Our first step is to assume a convenient parameterization for the matrices, $g_i(\mathbf{x})$, that compute arc lengths. We then show how to fit these matrices, along with the parameters λ_i and a_{ij} , by maximum likelihood estimation.

A variety of parameterizations can be considered for the matrices, $g_i(\mathbf{x})$. In this paper, we consider the very simple form:

$$g_i(\mathbf{x}) = |(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)|^2 \sigma_i^{-1}, \quad (13)$$

where the parameters $\boldsymbol{\mu}_i$, Σ_i and σ_i are set by maximum likelihood estimation. Here, Σ_i and σ_i are positive-definite $D \times D$ square matrices, while $\boldsymbol{\mu}_i$ is a D -dimensional vector. We also impose the determinant constraint $|\Sigma_i| |\sigma_i|^{\frac{1}{2}} = 1$; this eliminates the degenerate solution, $g_i(\mathbf{x}) = 0$, in which every trajectory is assigned zero arc length. Note that there remains an artificial degree of freedom associated with simultaneously rescaling Σ_i and σ_i .

The form of eq. (13) is designed to endow each state with a characteristic signature. In particular, consider the differential arc lengths that appear in eq. (6):

$$[\dot{\mathbf{x}}^T g_i(\mathbf{x}) \dot{\mathbf{x}}]^{\frac{1}{2}} = (\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) [\dot{\mathbf{x}}^T \sigma_i^{-1} \dot{\mathbf{x}}]^{\frac{1}{2}}.$$

If \mathbf{x} is close to $\boldsymbol{\mu}_i$, then both the arc length and the corresponding probability of decay (out of state i) are small. Each state is therefore characterized by the values of \mathbf{x} that allow it to persist. Intuitively, the parameters $\boldsymbol{\mu}_i$ can be viewed as *target* vectors associated with each state of the Markov process. Typical deviations about $\boldsymbol{\mu}_i$ are encoded by Σ_i and σ_i . In what follows, we show how to learn the parameters that best characterize each state.

3.1 LABELED EXAMPLES

Suppose we are given examples of segmented trajectories, $\{\mathbf{x}_\alpha(t), s_\alpha(t)\}$, where the index α runs over the example in the training set. As shorthand, let $\delta_{i\alpha}(t)$ denote the indicator function that selects out segments associated with state i :

$$\delta_{i\alpha}(t) = \begin{cases} 1 & \text{if } s_\alpha(t) = i, \\ 0 & \text{otherwise.} \end{cases} \quad (14)$$

Also, let $\ell_{i\alpha}$ denote the total arc length traversed by state i in the α th example:

$$\ell_{i\alpha} = \int dt \delta_{i\alpha}(t) [\dot{\mathbf{x}}_\alpha^T g_i(\mathbf{x}_\alpha) \dot{\mathbf{x}}_\alpha]^{\frac{1}{2}}. \quad (15)$$

In this paper we view learning as a problem in maximum likelihood estimation. Thus we seek the parameters that maximize the conditional log-likelihood:

$$\sum_\alpha \ln \Pr[s_\alpha | \mathbf{x}_\alpha] = - \sum_{i\alpha} \lambda_i \ell_{i\alpha} + \sum_{ij} n_{ij} \ln[\lambda_i a_{ij}], \quad (16)$$

where n_{ij} is the overall number of observed transitions from state i to state j . The first term in eq. (16) measures the log-likelihood of observed segments in isolation, while the second measures the log-likelihood of observed transitions.

Eq. (16) has a convenient form for maximum likelihood estimation. In particular, there are closed-form solutions for the values of λ_i and a_{ij} that maximize this log-likelihood; they are given by:

$$a_{ij} = n_{ij} / n_i, \quad (17)$$

$$\lambda_i^{-1} = \frac{1}{n_i} \sum_\alpha \ell_{i\alpha}, \quad (18)$$

where $n_i = \sum_j n_{ij}$. In general, we cannot find closed-form solutions for the maximum-likelihood estimates of $\{\boldsymbol{\mu}_i, \Sigma_i, \sigma_i\}$. However, we can update these parameters in an iterative fashion that is guaranteed to increase the log-likelihood at each step. Denoting the updated parameters by $\{\tilde{\boldsymbol{\mu}}_i, \tilde{\Sigma}_i, \tilde{\sigma}_i\}$, we consider the iterative scheme (derived in the appendix):

$$\tilde{\boldsymbol{\mu}}_i \leftarrow \frac{\sum_\alpha \int dt \delta_{i\alpha} [\dot{\mathbf{x}}_\alpha^T \sigma_i^{-1} \dot{\mathbf{x}}_\alpha]^{\frac{1}{2}} \mathbf{x}_\alpha}{\sum_\alpha \int dt \delta_{i\alpha} [\dot{\mathbf{x}}_\alpha^T \sigma_i^{-1} \dot{\mathbf{x}}_\alpha]^{\frac{1}{2}}}, \quad (19)$$

$$\tilde{\Sigma}_i \leftarrow \frac{\sum_\alpha \int dt \delta_{i\alpha} [\dot{\mathbf{x}}_\alpha^T \sigma_i^{-1} \dot{\mathbf{x}}_\alpha]^{\frac{1}{2}} \Delta_{i\alpha} \Delta_{i\alpha}^T}{\sum_\alpha \int dt \delta_{i\alpha} [\dot{\mathbf{x}}_\alpha^T \sigma_i^{-1} \dot{\mathbf{x}}_\alpha]^{\frac{1}{2}}}, \quad (20)$$

$$\tilde{\sigma}_i \leftarrow c_i \sum_\alpha \int dt \delta_{i\alpha} \frac{\Delta_{i\alpha}^T \tilde{\Sigma}_i^{-1} \Delta_{i\alpha}}{[\dot{\mathbf{x}}_\alpha^T \sigma_i^{-1} \dot{\mathbf{x}}_\alpha]^{\frac{1}{2}}} \dot{\mathbf{x}}_\alpha \dot{\mathbf{x}}_\alpha^T, \quad (21)$$

where the constant c_i is determined by the determinant constraint $|\tilde{\Sigma}_i| |\tilde{\sigma}_i|^{\frac{1}{2}} = 1$ and we have introduced the shorthand notation,

$$\Delta_{i\alpha}(t) = \mathbf{x}_\alpha(t) - \tilde{\boldsymbol{\mu}}_i, \quad (22)$$

for the difference between $\mathbf{x}_\alpha(t)$ and its (re-estimated) target value in state i . Note that all the variables in eqs. (19–21) with the subscript α have an implicit time dependence.

Some intuition for the form of these updates can be gained by considering the points distributed along $\mathbf{x}_\alpha(t)$, as weighted by the measure $\delta_{i\alpha}(t) [\dot{\mathbf{x}}^T \sigma_i^{-1} \dot{\mathbf{x}}]^{\frac{1}{2}}$. The updates for $\boldsymbol{\mu}_i$ and Σ_i simply compute the mean and covariance of this distribution. The update for σ_i has a similar interpretation, though its derivation relies on the introduction of an auxiliary function, $Q(\tilde{\sigma}_i, \sigma_i)$, as in the Expectation-Maximization (EM) procedure (Dempster, Laird, & Rubin, 1977). Note that it is important to perform the updates in the order shown, since (for example) the $\tilde{\Sigma}$ -update depends on the re-estimated value of $\tilde{\boldsymbol{\mu}}$. By taking gradients of eq. (16), one can show that the fixed points of this iterative procedure correspond to stationary points of the log-likelihood. A proof sketch of monotonic convergence is given in the appendix.

In the case of labeled examples, the above procedures for maximum likelihood estimation can be invoked independently for each state i . One first iterates eqs. (19–21) to estimate the parameters that determine $g_i(\mathbf{x})$. These parameters are then used to compute the arc lengths, $\ell_{i\alpha}$, that appear in eq. (15). Given these arc lengths, the decay parameters and transition probabilities follow directly from eqs. (17–18). Thus the problem of learning given labeled examples is relatively straightforward.

3.2 UNLABELED EXAMPLES

In this section we consider the problem of unsupervised learning. In this setting, the learner does not have access to labeled examples; the only available information consists of the trajectories $\mathbf{x}_\alpha(t)$, as well as the fact that each process terminates at some time τ_α . The goal of unsupervised learning is to maximize the conditional log-likelihood,

$$\sum_{\alpha} \ln \Pr[s_\alpha(\tau_\alpha) = \text{END} | \mathbf{x}_\alpha(t)], \quad (23)$$

that for each trajectory $\mathbf{x}_\alpha(t)$, some probable segmentation can be found that terminates at precisely the observed time. The marginal probabilities in eq. (23)

are computed by summing $\Pr[s(t) | \mathbf{x}(t)]$ over allowed segmentations, as in eq. (12).

The maximization of this log-likelihood defines a problem in hidden variable density estimation. The hidden variables are the states of the Markov process. If these variables were known, the problem would reduce to the one considered in the previous section. To fill in these missing values, we avail ourselves of the Expectation-Maximization (EM) algorithm (Baum, 1972; Dempster, Laird, & Rubin, 1976). Roughly speaking, the EM algorithm works by converting the maximization of eq. (23) into a weighted version of the problem where the segmentations, $s_\alpha(t)$, are known. The weights are determined by the posterior probabilities, $\Pr[s_\alpha(t) | \mathbf{x}_\alpha(t), s_\alpha(\tau_\alpha) = \text{END}]$, derived from the current parameter estimates.

In the interest of brevity, we do not give a detailed account of the full EM algorithm for MPCs. We note, however, that eqs. (10–11) by themselves suffice to implement a very good approximation to the full procedure. This approximation is to compute, based on the current parameter estimates, the optimal segmentation, $s_\alpha^*(t)$, for each trajectory in the training set; one then re-estimates the parameters of the Markov process by treating the inferred segmentations, $s_\alpha^*(t)$, as targets. This approximation reduces the problem of parameter estimation to the one considered in the previous section. It can be viewed as a winner-take-all approximation to the full EM algorithm, analogous to the Viterbi approximation for hidden Markov models (Rabiner & Juang, 1993).

Essentially the same algorithm can also be applied to the intermediate case of *partially labeled* examples. Suppose, for example, that the learner has access to labeled state sequences but not to segmented curves; in other words, examples are provided in the form:

$$\{\text{START} \rightarrow (s_1, ?) \cdots (s_n, ?) \rightarrow \text{END}\}. \quad (24)$$

The ability to handle such examples is important for two reasons: first, because they provide significantly more information than unlabeled examples, and second, because they are often much cheaper to generate than fully segmented curves. As before, we can view the learning problem for these examples as one in hidden variable density estimation. In this case, the hidden variables are not the states of the Markov process per se, but only the times at which they change. We can incorporate knowledge of the state sequence into the EM algorithm simply by restricting the sums over paths in eqs. (10) and (12) to those that pass through the desired sequence.

4 AUTOMATIC SPEECH RECOGNITION

The Markov processes in this paper were conceived as models for automatic speech recognition (Rabiner & Juang, 1993). Speech recognizers take as input a sequence of feature vectors, each of which encodes a short window of speech. Acoustic feature vectors typically have ten or more components, so that a particular sequence of feature vectors can be viewed as tracing out a multidimensional curve. The goal of a speech recognizer is to translate this curve into a sequence of words, or more generally, a sequence of sub-syllabic units known as *phonemes*. Denoting the feature vectors by \mathbf{x}_t and the phonemes by s_t , we can view this problem as the discrete-time equivalent of the segmentation problem in MPCs.

Why consider MPCs as models of speech recognition? Hidden Markov models (HMMs), the current leading technology, are also based on probabilistic methods. These models manipulate joint distributions of the form:

$$\Pr[s, \mathbf{x}] = \prod_t \Pr[s_t | s_{t-1}] \Pr[\mathbf{x}_t | s_t]. \quad (25)$$

Though HMMs have led to significant advances in speech recognition, they are handicapped by certain weaknesses. One of these is the poor manner in which they handle variations in speaking rate. Intuitively, we can represent these variations by nonlinear warpings of time. For example, consider the pair of trajectories \mathbf{x}_t and \mathbf{y}_t , where \mathbf{y}_t is created by the doubling operation:

$$\mathbf{y}_t = \begin{cases} \mathbf{x}_{t/2} & \text{if } t \text{ even,} \\ \mathbf{y}_{t-1} & \text{if } t \text{ odd.} \end{cases} \quad (26)$$

Both trajectories trace out the same curve, but \mathbf{y}_t does so at half the rate as \mathbf{x}_t . Hidden Markov models will not assign these trajectories the same likelihood, nor are they guaranteed to infer equivalent segmentations. This example shows that HMMs do not even approximately capture the invariances modeled by MPCs or other arc-length based descriptions of speech (Tishby, 1990).

Admittedly, the warping in eq. (26) represents a highly idealized picture of acoustic variability. Nevertheless, there is a great deal of empirical evidence that HMMs suffer from the inability to model variations in speaking rate (Siegler & Stern, 1995). For example, word error rates increase dramatically when one moves from scripted to spontaneous speech. Also, one generally observes that consonants are more frequently botched

than vowels. The reason is that in HMMs, the contribution of particular states to the overall log-likelihood is in direct proportion to their duration. Thus training procedures designed to maximize the log-likelihood are inherently biased to model long-lived phonemes (i.e., vowels) more accurately than short-lived ones.

MPCs are quite different from HMMs in this respect. In MPCs, the contribution of each state to the log-likelihood is determined by its arc length. The weighting by arc length attaches a more important role to short-lived but *non-stationary* phonemes. Of course, one can imagine heuristics in HMMs that achieve the same effect, such as dividing each state's contribution to the log-likelihood by its observed (or inferred) duration. Unlike such heuristics, however, the state-dependent metric $g(\mathbf{x})$ in MPCs is *learned* from data; in particular, it is designed to reweight the speech signal in a way that reflects the actual statistics of acoustic trajectories.

So far we have emphasized the invariance to nonlinear warpings of time as the main difference between MPCs and HMMs. Another important difference, however, lies in what each tries to model. While MPCs attempt to model the conditional distribution $\Pr[s|\mathbf{x}]$, HMMs attempt to model the joint distribution, $\Pr[s, \mathbf{x}]$. Only the former is required for speech recognition, yet HMMs attempt something much more ambitious by learning a *generative* model of acoustic trajectories. Maximum likelihood training in HMMs is designed to increase the likelihood of observed trajectories, $\Pr[\mathbf{x}]$. Unfortunately, because HMMs do not represent the true model of speech, maximizing this likelihood does not always translate into minimizing error rates. These issues point to yet another difference between MPCs and HMMs. Learning in MPCs is directed at learning a *recognition* model, $\Pr[s|\mathbf{x}]$, as opposed to a *synthesis* model, $\Pr[\mathbf{x}|s]$. The direction of conditioning is a crucial difference between maximum likelihood estimation in MPCs and HMMs.

In terms of previous work, our motivation for MPCs most closely resembles that of Tishby (1990), who stressed the importance of invariance to nonlinear warpings of time as a mathematical symmetry. In that MPCs stress the continuous nature of the speech signal, they also bear some resemblance to so-called *segmental acoustic models* (Ostendorf, Digalakis, & Kimball, 1996) of speech. Unlike HMMs, segmental acoustic models enforce the constraint that acoustic feature vectors within the same phonemic state trace out a continuous trajectory. Despite this shared emphasis on continuity, however, segmental models and MPCs

differ in fundamental respects. In particular, segmental models incorporate the constraint of continuity by building a more complicated synthesis model $\Pr[\mathbf{x}|s]$ of acoustic trajectories. They retain, however, the usual Markov assumption between states:

$$\Pr[s_t|s_{t-1}, s_{t-2}, \dots, s_{t-\tau}] = \Pr[s_t|s_{t-1}]. \quad (27)$$

By contrast, MPCs build a recognition model $\Pr[s|\mathbf{x}]$ whose very definition is conditioned on the existence of a continuous trajectory. Moreover, the Markov assumption in MPCs—as embodied by eq. (6)—is conditioned on the current position and tangent vector of the acoustic feature trajectory. This differs from the Markov assumption in eq. (27), which is made independent of (or unconditioned on) the acoustic features. Finally, to the best of our knowledge, MPCs are novel in two key respects: the formulation of a warp-invariant probabilistic model explicitly in terms of arc length, and the emphasis on learning a metric $g(\mathbf{x})$ for each hidden state of the Markov process. These ideas differentiate MPCs from segmental acoustic models as well as ordinary HMMs.

The starting point of this work was to postulate eq. (1) as an invariance of random processes. Of course, it would be naive to expect speech signals to exhibit a *strict* invariance to nonlinear warpings of time. The acoustic realization of a phoneme does depend to some extent on the speaking rate, and certain phonemes are more likely to be stretched or shortened than others. To accommodate this, one can relax the warping invariance in MPCs. This is most easily done by building models of the *space-time*¹ trajectories $\mathbf{X}(t) = \{\mathbf{x}(t), t\}$ and computing generalized arc lengths, $dL = [\dot{\mathbf{X}}^T G(\mathbf{X}) \dot{\mathbf{X}}]^{\frac{1}{2}} dt$, where $\dot{\mathbf{X}} = \{\dot{\mathbf{x}}, 1\}$ and $G(\mathbf{X})$ is a space-time metric. The effect of replacing $\dot{\mathbf{x}}$ by $\dot{\mathbf{X}}$ is to allow each acoustic feature vector to contribute a finite amount to the overall log-likelihood even when $|\dot{\mathbf{x}}|$ is zero—that is, even when it represents a perfectly stationary frame of speech.

We are currently evaluating MPCs as engines for automatic speech recognition. Naturally, we expect that many further elaborations will be required to surpass the finely tuned performance of modern recognizers. These may include more sophisticated parameterizations of the metric $g_i(\mathbf{x})$, the use of information from higher order derivatives (e.g., $\dot{\mathbf{x}}$ and $\ddot{\mathbf{x}}$), and/or transition probabilities $a_{ij}(\mathbf{x})$ that vary along the length

¹The admixture of space and time coordinates in this way is an old idea from physics, originating in the theory of relativity (Einstein, 1924) (though in that context the metric is negative-definite).

of the curve. Nevertheless, we hope that this paper serves to introduce the basic principles of MPCs, as well as to suggest an intriguing departure from traditional methods in automatic speech recognition.

A REESTIMATION FORMULAS

In this appendix we derive the reestimation formulas, eqs. (19–21) and show that they lead to monotonic increases in the log-likelihood, eq. (16).

We begin by examining a simpler problem. Let $\{\mathbf{x}(t)|t \in [0, \tau]\}$ denote a D -dimensional trajectory, and let $\Phi(\mathbf{x}) \geq 0$ denote an everywhere non-negative function of \mathbf{x} . Now consider the function:

$$\ell(\sigma) = \int_0^\tau dt \left[\dot{\mathbf{x}}^T \sigma^{-1} \dot{\mathbf{x}} \right]^{\frac{1}{2}} \Phi(\mathbf{x}(t)), \quad (28)$$

where σ is a $D \times D$ positive-definite matrix. The right hand side of eq. (28) clearly depends on the trajectory $\mathbf{x}(t)$ and the function $\Phi(\mathbf{x})$, but for now let us regard both of these as fixed and consider $\ell(\sigma)$ simply as a function of the matrix σ .

Since σ is positive-definite and $\Phi(\mathbf{x}) \geq 0$, we immediately observe that the function $\ell(\sigma)$ is bounded below by zero. Let us consider how to find the value of σ that minimizes $\ell(\sigma)$, subject to the determinant constraint $|\sigma| = 1$. Note that the matrix elements of σ^{-1} appear nonlinearly in the right hand side of eq. (28); thus it is not possible to compute their optimal values in closed form. As an alternative, we consider the auxiliary function:

$$Q(\rho, \sigma) = \int_0^\tau dt \left\{ \frac{\dot{\mathbf{x}}^T \rho^{-1} \dot{\mathbf{x}}}{[\dot{\mathbf{x}}^T \sigma^{-1} \dot{\mathbf{x}}]^{\frac{1}{2}}} + [\dot{\mathbf{x}}^T \sigma^{-1} \dot{\mathbf{x}}]^{\frac{1}{2}} \right\} \frac{\Phi(\mathbf{x}(t))}{2}, \quad (29)$$

where ρ is a $D \times D$ positive-definite matrix like σ . It follows directly from the definition in eq. (29) that $\ell(\sigma) = Q(\sigma, \sigma)$. Somewhat less trivially, we observe that $Q(\rho, \rho) \leq Q(\rho, \sigma)$ for all positive definite matrices ρ and σ . This inequality follows from the concavity of the square root function, as illustrated in figure 2.

Consider the value of ρ which minimizes $Q(\rho, \sigma)$, subject to the determinant constraint $|\rho| = 1$. We denote this value by $\tilde{\sigma} = \min_{|\rho|=1} Q(\rho, \sigma)$. Because the matrix elements of ρ^{-1} appear *linearly* in $Q(\rho, \sigma)$, this minimization essentially reduces to computing the covariance matrix of the tangent vector $\dot{\mathbf{x}}$, as distributed along the trajectory $\mathbf{x}(t)$. In particular, we have:

$$\tilde{\sigma} \propto \int_0^\tau dt \frac{\dot{\mathbf{x}} \dot{\mathbf{x}}^T}{[\dot{\mathbf{x}}^T \sigma^{-1} \dot{\mathbf{x}}]^{\frac{1}{2}}} \Phi(\mathbf{x}(t)), \quad (30)$$

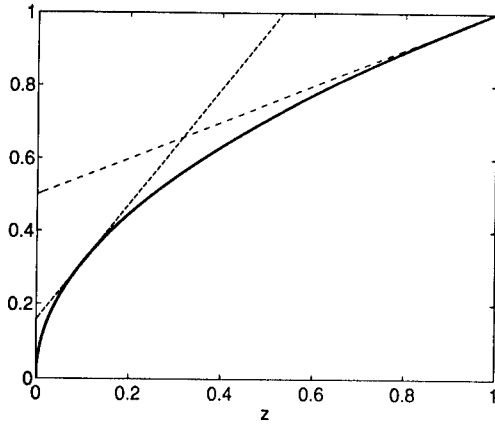


Figure 2: The square root function is concave and upper bounded by $\sqrt{z} \leq \frac{1}{2}[z/\sqrt{\xi} + \sqrt{\xi}]$ for all $\xi \geq 0$. The bounding tangents are shown for $\xi = \frac{1}{10}$ and $\xi = 1$.

where the constant of proportionality is determined by the constraint $|\tilde{\sigma}| = 1$. To minimize $\ell(\sigma)$ with respect to σ , we now consider the iterative procedure where at each step we replace σ by $\tilde{\sigma}$. We observe that:

$$\begin{aligned} \ell(\tilde{\sigma}) &= Q(\tilde{\sigma}, \tilde{\sigma}) \\ &\leq Q(\tilde{\sigma}, \sigma) \text{ due to concavity} \\ &\leq Q(\sigma, \sigma) \text{ since } \tilde{\sigma} = \min_{\rho} Q(\rho, \sigma) \\ &= \ell(\sigma), \end{aligned}$$

with equality generally holding only when $\tilde{\sigma} = \sigma$. In other words, this iterative procedure converges *monotonically* to a local minimum of $\ell(\sigma)$.

Let us now relate the problem of minimizing $\ell(\sigma)$ to the original problem of maximizing the likelihood in eq. (16). There we saw that for each state of the MPC, it was necessary to optimize the parameters $\{\mu, \Sigma, \sigma\}$. Here, for notational convenience, we have dropped the subscript denoting the state index of these parameters. Note that in terms of these parameters, maximizing each state's contribution to the log-likelihood is equivalent to minimizing the total arc length of its segments in the training set. This problem can be viewed as a particular instance of the one considered above, provided that we make the identification:

$$\Phi(\mathbf{x}) = (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu). \quad (31)$$

Of course, now in addition to minimizing the arc length with respect to σ , we must also optimize the values of μ and Σ . To this end, note that eq. (31) defines a standard quadratic form; hence for fixed σ , the values of μ and Σ that minimize eq. (28) are given simply by the mean and covariance matrix of the points $\mathbf{x}(t)$

along each state's segments, as weighted by the measure $[\dot{\mathbf{x}}^T \sigma^{-1} \dot{\mathbf{x}}]^{\frac{1}{2}}$. Within each state, we thus obtain a monotonically convergent learning procedure by alternately optimizing μ and Σ for fixed σ , then optimizing σ for fixed μ and Σ . This leads directly to the reestimation formulas in eqs. (19–21).

Acknowledgements

The author thanks F. Pereira, M. Rahim, and the anonymous reviewers for many helpful comments about the presentation of these ideas.

References

- L. Baum (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of a markov process. In O. Shisha, editor, *Inequalities*, 3:1–8. New York: Academic Press.
- A. Dempster, N. Laird, and D. Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society B*, 39:1–38.
- M. P. do Carmo (1976) *Differential Geometry of Curves and Surfaces*. Prentice Hall.
- R. O. Duda and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.
- A. Einstein (1924). *The Principle of Relativity*. Dover.
- M. Ostendorf, V. Digalakis, and O. Kimball (1996). From HMMs to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 4:360–378.
- L. Rabiner and B. Juang (1993). *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice Hall.
- M. A. Siegler and R. M. Stern (1995). On the effects of speech rate in large vocabulary speech recognition systems. In *Proceedings of the 1995 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 612–615.
- P. Simard, Y. LeCun, and J. Denker (1993). Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems* 5:50–58. San Mateo, CA: Morgan Kaufman.
- N. Tishby (1990). A dynamical system approach to speech processing. In *Proceedings of the 1990 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 365–368.

Ridge Regression Learning Algorithm in Dual Variables

C. Saunders, A. Gammerman and V. Vovk

Royal Holloway, University of London

Egham, Surrey, TW20 0EX, UK

{craig,alex,vovk}@dcs.rhnc.ac.uk

Abstract

In this paper we study a dual version of the Ridge Regression procedure. It allows us to perform non-linear regression by constructing a linear regression function in a high dimensional feature space. The feature space representation can result in a large increase in the number of parameters used by the algorithm. In order to combat this "curse of dimensionality", the algorithm allows the use of kernel functions, as used in Support Vector methods. We also discuss a powerful family of kernel functions which is constructed using the ANOVA decomposition method from the kernel corresponding to splines with an infinite number of nodes. This paper introduces a regression estimation algorithm which is a combination of these two elements: the dual version of Ridge Regression is applied to the ANOVA enhancement of the infinite-node splines. Experimental results are then presented (based on the Boston Housing data set) which indicate the performance of this algorithm relative to other algorithms.

1 INTRODUCTION

First of all, let us formulate regression estimation problem. Suppose we have a set of vectors¹ x_1, \dots, x_T , and we also have a supervisor which gives us a real value y_t , for each of the given vectors. Our problem is to construct a learning machine which when given a new

set of examples, minimises some measure of discrepancy between its prediction \hat{y}_t and the value of y_t . The measure of loss which we are using, average square loss (L), is defined by

$$L = \frac{1}{l} \sum_{t=1}^l (y_t - \hat{y}_t)^2,$$

where y_t are the supervisor's answers, \hat{y}_t are the predicted values, and l is the number of vectors in the test set.

Least Squares and Ridge Regression are classical statistical algorithms which have been known for a long time. They have been widely used, and recently some papers such as Drucker *et al.* [2] have used regression in conjunction with a high dimensional feature space. That is the original input vectors are mapped into some feature space, and the algorithms are then used to construct a linear regression function in the feature space, which represents a non-linear regression in the original input space. There is, however, a problem encountered when using these algorithms within a feature space. Very often we have to deal with a very large number of parameters, and this leads to serious computational difficulties that can be impossible to overcome. In order to combat this "curse of dimensionality" problem, we describe a dual version of the Least Squares and Ridge Regression algorithms, which allows the use of kernel functions. This approach is closely related to Vapnik's kernel method as used in the Support Vector Machine. Kernel functions represent dot products in a feature space, which allows the algorithms to be used in a feature space without having to carry out computations within that space. Kernel functions themselves can take many forms and particular attention is paid to a family of kernel functions which are constructed using ANOVA decomposition (Vapnik [10]; see also Wahba [11, 12]). There are two

¹We will use subscripts to indicate a particular vector (e.g. x_t is the t th vector), and superscripts to indicate a particular vector element (e.g. x^i is the i th element of the vector x).

major objectives of this paper:

1. To show how to use kernel functions to overcome the curse of dimensionality in the above mentioned algorithms.
2. To demonstrate how ANOVA decomposition kernels can be constructed, and evaluate their performance compared to polynomial and spline kernels, on a real world data set.

Results from experiments performed on the well known Boston housing data set are then used to show that the Least Squares and Ridge Regression algorithms perform well in comparison with some other algorithms. The results also show that the ANOVA kernels, which only consider a subset of the input parameters, can improve on results obtained on the same kernel function without the ANOVA technique applied. In the next section we present the dual form of Least Squares and Ridge Regression.

2 RIDGE REGRESSION IN DUAL VARIABLES

Before presenting the algorithms in dual variables, the original formulation of Least Squares and Ridge Regression is stated here for clarity.

Suppose we have a training set $(x_1, y_1), \dots, (x_T, y_T)$, where T is the number of examples, x_t are vectors in \mathbb{R}^n (n is the number of attributes) and $y_t \in \mathbb{R}$, $t = 1, \dots, T$. Our comparison class consists of the linear functions $y = w \cdot x$, where $w \in \mathbb{R}^n$.

The Least Squares method recommends computing $w = w_0$ which minimizes

$$L_T(w) = \sum_{t=1}^T (y_t - w \cdot x_t)^2$$

and using w_0 for labeling future examples: if a new example has attributes x , the predicted label is $w_0 \cdot x$.

The Ridge Regression procedure is a slight modification on the least squares method and replaces the objective function $L_T(w)$ by

$$a\|w\|^2 + \sum_{t=1}^T (y_t - w \cdot x_t)^2,$$

where a is a fixed positive constant.

We now derive a "dual version" for Ridge Regression (RR); since we allow $a = 0$, this includes Least Squares

(LS) as a special case. In this derivation we partially follow Vapnik [8]. We start with re-expressing our problem as: minimize the expression

$$a\|w\|^2 + \sum_{t=1}^T \xi_t^2 \quad (1)$$

under the constraints

$$y_t - w \cdot x_t = \xi_t, \quad t = 1, \dots, T. \quad (2)$$

Introducing Lagrange multipliers α_t , $t = 1, \dots, T$, we can replace our constrained optimization problem by the problem of finding the saddle point of the function

$$a\|w\|^2 + \sum_{t=1}^T \xi_t^2 + \sum_{t=1}^T \alpha_t (y_t - w \cdot x_t - \xi_t). \quad (3)$$

In accordance with the Kuhn—Tucker theorem, there exist values of Lagrange multipliers $\alpha = \alpha^{KT}$ for which the minimum of (3) equals the minimum of (1), under constraints (2). To find the optimal w and ξ , we will do the following; first, minimize (3) in w and ξ and then maximize it in α . Notice that for any fixed values of α the minimum of (3) (in w and ξ) is less than or equal to the value of the optimization problem (1)–(2), and equality is attained when $\alpha = \alpha^{KT}$. By doing this, we will therefore find the solution to our original constrained minimization problem (1)–(2).

Differentiating (3) in w , we obtain the condition

$$2aw - \sum_{t=1}^T \alpha_t x_t = 0,$$

i.e.,

$$w = \frac{1}{2a} \sum_{t=1}^T \alpha_t x_t. \quad (4)$$

(Lagrange multipliers are usually interpreted as reflecting the importance of the corresponding constraints, and equation (4) shows that w is proportional to the linear combination of x_t , each of which is taken with a weight proportional to its importance.) Substituting this into (3), we obtain

$$\begin{aligned} & \frac{1}{4a} \sum_{s,t=1}^T \alpha_s \alpha_t (x_s \cdot x_t) + \sum_{t=1}^T \xi_t^2 \\ & + \frac{1}{2a} \left(\sum_{t=1}^T \alpha_t x_t \right) \cdot \left(- \sum_{t=1}^T \alpha_t x_t \right) + \sum_{t=1}^T y_t \alpha_t - \sum_{t=1}^T \alpha_t \xi_t \end{aligned}$$

$$= -\frac{1}{4a} \sum_{s,t=1}^T \alpha_s \alpha_t (x_s \cdot x_t) + \sum_{t=1}^T \xi_t^2 + \sum_{t=1}^T y_t \alpha_t - \sum_{t=1}^T \alpha_t \xi_t. \quad (5)$$

Differentiating (5) in ξ_t , we obtain

$$\xi_t = \frac{\alpha_t}{2}, \quad t = 1, \dots, T \quad (6)$$

(i.e., the importance of the t th constraint is proportional to the corresponding residual); substitution into (5) gives

$$-\frac{1}{4a} \sum_{s,t=1}^T \alpha_s \alpha_t (x_s \cdot x_t) - \frac{1}{4} \sum_{t=1}^T \alpha_t^2 + \sum_{t=1}^T y_t \alpha_t. \quad (7)$$

Denoting K as the $T \times T$ matrix of dot products

$$K_{s,t} = x_s \cdot x_t,$$

and differentiating in α_t , we obtain the condition

$$-\frac{1}{2a} K \alpha - \frac{1}{2} \alpha + y = 0,$$

which is equivalent to

$$\alpha = 2a(K + aI)^{-1}y.$$

Recalling (4), we obtain that the prediction y given by the Ridge Regression procedure on the new unlabeled example x is

$$w \cdot x = \left(\frac{1}{2a} \sum_{t=1}^T \alpha_t x_t \right) \cdot x = \frac{1}{2a} \alpha \cdot k = y'(K + aI)^{-1}k,$$

where $k = (k_1, \dots, k_T)'$ is the vector of the dot products:

$$k_t := x_t \cdot x, \quad t = 1, \dots, T.$$

Lemma 1 *RR's prediction of the label y of a new unlabeled example x is*

$$y'(K + aI)^{-1}k, \quad (8)$$

where K is the matrix of dot products of the vectors x_1, \dots, x_T in the training set,

$$K_{s,t} = K(x_s, x_t), \quad s = 1, \dots, T, \quad t = 1, \dots, T,$$

k is the vector of dot products of x and the vectors in the training set,

$$k_t := K(x_t, x), \quad t = 1, \dots, T,$$

and $K(x, x') = x \cdot x'$ is simply a function which returns the dot product of the two vectors, x and x' .

3 LINEAR REGRESSION IN FEATURE SPACE

When $K(x_i, x_j)$ is simply a function which returns the dot product of the given vectors, formula (8) corresponds to performing linear regression within the input space \mathbb{R}^n defined by the examples. If we want to construct a linear regression in some feature space, we first have to choose a mapping from the original space X to a higher dimensional feature space F ($\phi: X \rightarrow F$). In order to use Lemma 1 to construct the regression in the feature space, the function K must now correspond to the dot product $\phi(x_i) \cdot \phi(x_j)$. It is not necessary to know $\phi(x)$ as long as we know $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. The question of which functions K correspond to a dot product in some feature space F is answered by Mercer's theorem and addressed by Vapnik [9] in his discussion of support vector methods. As an illustration of the idea, an example of a simple kernel function is presented here. (See Girosi [4].) Suppose there is a mapping function ϕ which maps a two-dimensional vector into 6 dimensions:

$$\phi: (x^1, x^2) \mapsto ((x^1)^2, (x^2)^2, \sqrt{2}x^1, \sqrt{2}x^2, \sqrt{2}x^1x^2, 1),$$

then dot products in F take the form

$$\begin{aligned} & (\phi(x) \cdot \phi(y)) \\ &= (x^1)^2(y^1)^2 + (x^2)^2(y^2)^2 + 2x^1y^1 \\ & \quad + 2x^2y^2 + 2x^1y^1x^2y^2 + 1 \\ &= ((x \cdot y) + 1)^2. \end{aligned}$$

One possible kernel function is therefore $((x \cdot y) + 1)^2$. This can be generalised into a kernel function of the form

$$K(x, y) = ((x \cdot y) + 1)^d,$$

and more than 2 dimensions.

The use of kernel functions allows us to construct a linear regression function in a high dimensional feature space (which corresponds to a non-linear regression in the input space) avoiding the curse of having to carry out computations in the high dimensional space. In particular, kernel functions are a way to combat the curse of dimensionality problems such as those faced in Drucker *et al.* [2], where a regression function was also constructed in a feature space, but computations were carried out in the high dimensional space, leading to huge number of parameters for non-trivial problems.

For more information on the kernel technique, see Vapnik [8, 10, 9] and Wahba [11].

4 MULTIPLICATIVE KERNELS

Before indicating how ANOVA decomposition can be used to form kernels, a brief description is needed of the family of kernels to which the ANOVA decomposition can be applied, this being the family of multiplicative kernels. This refers to the set of kernels where the multi-dimensional case is calculated as the product of the one-dimensional case. That is, if the one-dimensional case is $k(x^i, y^i)$, then the n -dimensional case is

$$\mathcal{K}_n(x, y) = \prod_{i=1}^n k(x^i, y^i).$$

One such kernel (to which the ANOVA decomposition is applied here) is the spline kernel with an infinite number of nodes (see Vapnik [8, 10] and Kimeldorf and Wahba [5]). A spline approximation which has an infinite number of nodes can be defined on the interval $(0, a)$, $0 < a < \infty$, as the expansion

$$f(x) = \int_0^a a(t)(x-t)_+^d dt + \sum_{i=0}^d a_i x^i,$$

where a_i , $i = 0, \dots, d$, are unknown values, and $a(t)$ is an unknown function which defines the expansion. This can be considered as an inner product, and the kernel which generates splines of dimension d with an infinite number of nodes can be expressed as

$$k_d(x, y) = \int_0^a (x-t)_+^d (y-t)_+^d dt + \sum_{r=0}^d x^r y^r.$$

Note that when $t > \min(x, y)$ the function under the integral sign will have value zero. It is therefore sufficient only to consider the interval $(0, \min(x, y))$, which makes the formula above equivalent to

$$k_d(x, y) = \sum_{r=0}^d \frac{\binom{d}{r}}{2d-r+1} \min(x, y)^{2d-r+1} |x-y|^r + \sum_{r=0}^d x^r y^r.$$

In particular, for the case of linear splines ($d = 1$) we have :

$$k_1(x, y) = 1 + xy + \frac{1}{2}|y-x|\min(x, y)^2 + \frac{\min(x, y)^3}{3}.$$

5 ANOVA DECOMPOSITION KERNELS

The ANOVA decomposition kernels are inspired by their namesake in statistics, which analyses different subsets of variables. The actual decomposition can be adapted to form kernels (as in, e.g., Vapnik [10]) which involve different subsets of the attributes of the examples up to a certain size. There are two main reasons for choosing to use ANOVA decomposition. Firstly, the different subsets which are considered may group together like variables, which can lead to greater predictive power. Also, by only considering some subsets of the input parameters, ANOVA decomposition reduces the VC dimension of the set of functions that you are considering, which can avoid overfitting your training data.

Given a one-dimensional kernel k , the ANOVA kernels are defined as follows:

$$\mathcal{K}_1(x, y) = \sum_{1 \leq k \leq n} k(x^k, y^k),$$

$$\mathcal{K}_2(x, y) = \sum_{1 \leq k_1 < k_2 \leq n} k(x^{k_1}, y^{k_1}) k(x^{k_2}, y^{k_2}),$$

$$\dots,$$

$$\mathcal{K}_n(x, y) = k(x^{k_1}, y^{k_1}) \dots k(x^{k_n}, y^{k_n}).$$

From Vapnik [10] the following recurrent procedure can be used when calculating the value of $\mathcal{K}_n(x, y)$. Let

$$\mathcal{K}^s(x, y) = \sum_{i=1}^n (k(x^i, y^i))^s$$

and $\mathcal{K}_0(x, y) = 1$; then

$$\mathcal{K}_p(x, y) = \sum_{1 \leq k_1 < k_2 < \dots < k_p \leq n} k(x^{k_1}, y^{k_1}) \dots k(x^{k_p}, y^{k_p}),$$

$$\mathcal{K}_p(x, y) = \frac{1}{p} \sum_{s=1}^p (-1)^{s+1} \mathcal{K}_{p-s}(x, y) \mathcal{K}^s(x, y).$$

For the purposes of this paper, when using kernels produced by ANOVA decomposition, only the order p is considered:

$$\mathcal{K}(x, y) = \mathcal{K}_p(x, y).$$

An alternative method of using ANOVA decomposition would be to consider order p and all lower orders (as in Stitson [7]), i.e.,

$$\mathcal{K}(x, y) = \sum_{i=1}^p \mathcal{K}_i(x, y).$$

6 EXPERIMENTAL RESULTS

Experiments were conducted on the Boston Housing data set². This is a well known data set for testing non-linear regression methods; see, e.g., Breiman [1] and Saunders [6]. The data set consists of 506 cases in which 12 continuous variables and 1 binary variable determine the median house price in a certain area of Boston in thousands of dollars. The continuous variables represent various values pertaining to different locational, economic and structural features of the house. The prices lie between \$5000 and \$50,000 in units of \$1000. Following the method used by Drucker *et al.* [2], the data set was partitioned into a training set of 401 cases, a validation set of 80 cases and a test set of 25 cases. This partitioning was carried out randomly 100 times, in order to carry out 100 trials on the data. For each trial the Ridge Regression algorithm was applied using:

- a kernel which corresponds to a spline approximation with an infinite number of nodes,
- the same kernel but with the ANOVA decomposition technique applied,
- and polynomial kernels.

For each kernel the set of parameters (the order of spline/degree of polynomial and the value of coefficient a) was selected which gave the smallest error on the validation set, and then the error on the test set was measured. This experiment was then repeated using a support vector machine (SVM), with the same kernels and exactly the same 100 training files (see Stitson [7] for full details). As an illustration of the number of parameters which were considered by the Ridge Regression Algorithm (and the SVM), consider the polynomial kernel which was outlined earlier, using a degree of 5. This maps the input vectors into a high dimensional feature space which is equivalent to evaluating $13^5 = 371,293$ different parameters.

The results obtained from the experiments are shown in Table 1. The measure of error used for the tests was the average squared error. For each of the 100 test files, the algorithm was run and the square of the difference between the predicted and actual value was taken. This was then averaged over the 25 test cases. This produces an average error for each of the 100 test

files, and an average of these were taken, which produces the final error which is quoted in the 3rd column of the table. The variance measure in the table is the average squared difference, between the squared error measured on each sample and the average squared error.

There are two additional results which should be noted here. One is from Breiman [1] using bagging with average squared error of 11.7, and one from Drucker *et al.* [2] using Support Vector regression with polynomial kernels with average squared error of 7.2. The result obtained by Drucker *et al.* is slightly better than the one obtained here using a similar machine; this may be, however, due to the random selection of the training, validation and testing sets.

7 COMPARISONS

In this section we will give a comparison of the results of this paper with the known results.

7.1 SV MACHINES

In this subsection we describe in more detail the connection of the approach of this paper with the Support Vector Machine.

Our optimization problem (minimizing (1) under constraints (2)) is essentially a special case of the following general optimization problem: minimize the expression

$$\frac{1}{2} \|w\|^2 + \frac{C}{k} \left(\sum_{t=1}^T (\xi_t^*)^k + \sum_{t=1}^T (\xi_t)^k \right) \quad (9)$$

under the constraints

$$y_t - w \cdot x_t \leq \epsilon + \xi_t^*, \quad t = 1, \dots, T, \quad (10)$$

$$w \cdot x_t - y_t \leq \epsilon + \xi_t, \quad t = 1, \dots, T; \quad (11)$$

$\epsilon > 0$ and $k \in \{1, 2\}$ are some constants. This optimization problem (along with a similar problem corresponding to Huber's loss function) is considered in Vapnik [10], Chapter 11 (Vapnik, however, considers more general regression functions of the form $w \cdot x + b$ rather than $w \cdot x$; the difference is minor because we can always add an extra attribute which is always 1 to all examples).

Our problem (1)–(2) corresponds to the problem (9)–(11) with $k = 2$, $\epsilon = 0$ and $C = 1/a$. Vapnik [10] gives a dual statement of his, and *a fortiori* our, problem; he does not reach, however, the closed-form expression (8)

²Available by anonymous FTP from:
ftp://ftp.ics.uci.com/pub/
machine-learning-databases/housing.

Table 1: Experimental Results on the Boston Housing Data

METHOD	KERNEL	SQUARED ERROR	VARIANCE
Ridge Regression	Polynomial	10.44	18.34
Ridge Regression	Splines	8.51	11.19
Ridge Regression	ANOVA Splines	7.69	8.27
SVM [7]	Polynomial	8.14	15.13
SVM	Splines	7.87	12.67
SVM	Anova Splines	7.72	9.44

(because he was mainly interested in positive values of ϵ).

As we mentioned before, our derivation of formula (8) follows [8]. The dual Ridge Regression is also known in traditional statistics, but statisticians usually use some clever matrix manipulations rather than the Lagrange method. Our derivation (modelled on Vapnik's) gives some extra insight: see, e.g., equations (4) and (6). For an excellent survey of connections between Support Vector Machine and the work done in statistics we refer the reader to Wahba [11, 12] and Girosi [4].

7.2 KRIEGING

Formula (8) is well known in the theory of Kriegering; in this subsection we will explain the connection for readers who are familiar with Kriegering. Consider the Bayesian setting where:

- the vector w of weights is distributed according to the normal distribution with mean 0 and covariance matrix $\frac{1}{2a}I$;
- $y_t = w \cdot x_t + \epsilon_t$, $t = 1, \dots, T$, where ϵ_t are random variables distributed normally with mean 0 and variance $\frac{1}{2}$.

Then the optimization problem (1) under the constraints (2) becomes the problem of finding the posterior mode (which, because of our normality assumption, coincides with the posterior mean) of w ; therefore, formula (8) gives the mean value of the random variable $w \cdot x$ (which is the "clean version" of the label $y = w \cdot x + \epsilon$ of the next example). Notice that the random variables $y_1, \dots, y_T, w \cdot x$ are jointly normal and the covariances between them are

$$\text{cov}(y_s, y_t) = \text{cov}(w \cdot x_s + \epsilon_s, w \cdot x_t + \epsilon_t) = \frac{1}{2a}(x_s \cdot x_t) + \frac{1}{2}$$

and

$$\text{cov}(y_t, w \cdot x) = \text{cov}(w \cdot x_t + \epsilon_t, w \cdot x) = \frac{1}{2a}(x_t \cdot x).$$

In accordance with the Kriegering formula the best prediction for $w \cdot x$ will be

$$y' \left(\frac{1}{2a}K + \frac{1}{2}I \right)^{-1} \left(\frac{1}{2a}k \right) = y'(K + aI)^{-1}k,$$

which coincides with (8).

8 CONCLUSIONS

A formula for Ridge Regression (which included Least Squares as a special case) in dual variables was derived using the method of Lagrange multipliers. This was then used to perform linear regression in a feature space. Therefore, we once more showed how the problem of learning in a very high dimensional space can be solved by using kernel functions. This allowed the algorithm to overcome the "curse of dimensionality" and run efficiently, even though a very large number of parameters were being considered. Experimental results show that Ridge Regression performs well. The results also indicate that applying ANOVA decomposition to a kernel can achieve better results than using the same kernel without the technique applied. Both Ridge Regression and the Support Vector method gave a smaller error when using ANOVA splines compared to the other spline kernel.

A weak part of our experimental section is that, though the Boston housing data is a useful benchmark, we have not applied our algorithm to a wider range of practical problems. This is what we plan to do next.

In order to confirm that ANOVA kernels can outperform kernels in their original form, the ANOVA decomposition technique should be applied to other multiplicative kernels. The technique of applying kernel functions to overcome problems of high dimensionality should also be investigated further, to see if it can be applied to any other algorithms which prove computationally difficult or impossible when faced with a large number of parameters.

We feel that a very interesting direction of developing the results of this paper would be to combine the dual version of Ridge Regression with the ideas of Gammerman *et al.* [3] to obtain a measure of confidence for predictions output by our algorithms. We expect that in this case simple closed-form formulas can be obtained.

Acknowledgments

We thank EPSRC for providing financial support through grant GR/L35812 ("Support Vector and Bayesian Learning Algorithms"). The referees' thoughtful comments are gratefully appreciated.

References

- [1] L. Breiman. Bagging predictors. Technical Report 421, Department of Statistics, University of California, Berkley, 1994. Also at: <ftp://ftp.stat.berkeley.edu/pub/tech-reports/421.ps.Z>.
- [2] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. N. Vapnik. Support Vector regression machines. In *Advances in Neural Information Processing Systems 9*, volume 9, page 155. The MIT Press, 1996.
- [3] A. Gammerman, V. Vapnik, and V. Vovk. Learning by transduction. In *Uncertainty in Artificial Intelligence*, 1998. To appear.
- [4] F. Girosi. An equivalence between sparse approximations and Support Vector Machines. Technical Report A. I. Memo No. 1606, C. B. C. L. Paper No. 147, Massachusetts Institute of Technology Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Department of Brain and Cognitive Sciences, May 1997.
- [5] G. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *J. Math. Anal. Appl.*, 33:82-95, 1971.
- [6] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression in dual variables. Technical report, Royal Holloway, University of London, 1998.
- [7] M. O. Stitson, A. Gammerman, V. N. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support Vector regression with ANOVA decomposition kernels. Technical report, Royal Holloway, University of London, 1997.
- [8] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
- [9] V. N. Vapnik. Statistical learning theory. In A. Gammerman, editor, *Computational Learning and Probabilistic Reasoning*. Wiley, 1996.
- [10] V. N. Vapnik. *Statistical Learning Theory*. Wiley, Forthcoming.
- [11] G. Wahba. *Spline models for observational data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1990.
- [12] G. Wahba. Support Vector machines, reproducing kernel Hilbert spaces and the randomized GACV. Technical Report 984, Department of Statistics, University of Wisconsin, USA, 1997.

Value Function Based Production Scheduling

Jeff G. Schneider*

Justin A. Boyan

Andrew W. Moore*

The Robotics Institute and Computer Science Department

Carnegie Mellon University

Pittsburgh, PA 15213

{schneide,jab,awm}@cs.cmu.edu

Abstract

Production scheduling, the problem of sequentially configuring a factory to meet forecasted demands, is a critical problem throughout the manufacturing industry. The requirement of maintaining product inventories in the face of unpredictable demand and stochastic factory output makes standard scheduling models, such as job-shop, inadequate. Currently applied algorithms, such as simulated annealing and constraint propagation, must employ ad-hoc methods such as frequent replanning to cope with uncertainty.

In this paper, we describe a Markov Decision Process (MDP) formulation of production scheduling which captures stochasticity in both production and demands. The solution to this MDP is a value function which can be used to generate optimal scheduling decisions online. A simple example illustrates the theoretical superiority of this approach over replanning-based methods. We then describe an industrial application and two reinforcement learning methods for generating an approximate value function on this domain. Our results demonstrate that in both deterministic and noisy scenarios, value function approximation is an effective technique.

production and demands, a Markov Decision Process (MDP) formulation is superior to the approaches currently in use. Our paper is organized as follows:

- Section 2 describes the abstract task of production scheduling and the sources of uncertainty which make the task difficult for current approaches. It also gives details of the particular scheduling instance we have worked on in collaboration with a major U.S. food manufacturer.
- Section 3 introduces the MDP model of the scheduling task and its solution based on value functions. A simple example illustrates that in the presence of uncertainty, the MDP model produces the optimal solution where both open-loop and closed-loop planners do not. We then discuss two reinforcement learning algorithms, Memory-based RTDP and ROUT, which are applicable for solving large-scale MDPs by value function approximation.
- Section 4 presents experimental results with ROUT and Memory-based RTDP on two somewhat simplified versions of the real-world manufacturing task. The results compare favorably to greedy and simulated annealing algorithms in both noisy and (surprisingly) deterministic scheduling scenarios.
- Finally, Section 5 discusses our results, related work, and promising future directions.

1 Introduction

Production scheduling is a critical problem throughout the manufacturing industry. In this paper, we argue that in order to deal with uncertainty in factory

* Also at Schenley Park Research, Inc.

2 Production Scheduling

2.1 Problem Specification

Production scheduling is the problem of deciding how to configure a factory sequentially to meet demands.

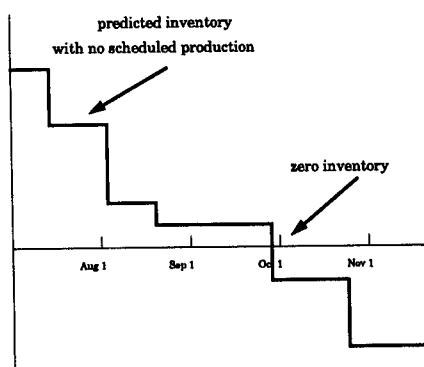


Figure 1: A demand curve for one product (see text for explanation)

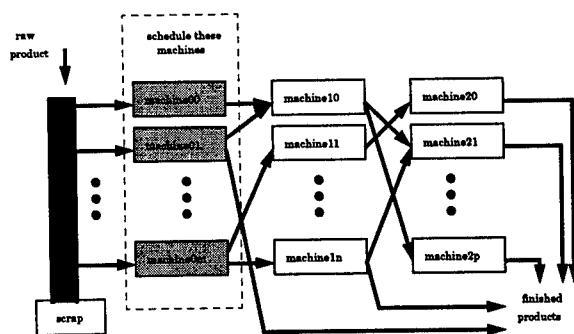


Figure 2: Factory layout (see text for explanation)

We restrict our attention here to a type of production scheduling called “make to stock.” We assume we have a modest number of products (2–100) and must produce enough of each to keep warehouse stocks high enough to satisfy customer requests for bulk shipments. This production model is common for most goods found in a supermarket. Automobile production, by contrast, is typically not scheduled under this model since cars are assembled individually with different options depending on specific customer orders.

An instance of the production scheduling problem is composed of five parts:

Machines and products. This is a list of what machines are present in the factory, and what products can be made on the machines. There may be complex constraints such as “machine A can only make product 1 when machine B is not making product 3.” A complete, legal assignment of products onto the set of machines is called a *configuration*. There is also a special “closed” configuration which represents a decision to shut the factory down.

Changeover times. It generally takes a certain amount of time to switch the factory from one configuration to another. During that time, there is no production. The problem definition includes a (possibly stochastic) estimate of how long it takes to change each configuration to each other configuration.

Production rates. Each configuration produces a set of products at a certain rate. There may be dependencies between the machines. For example, machine B may produce product 2 faster if machine A is also producing product 2. The actual production rates in the factory may be very stochastic; for example, some machines may jam frequently, causing irregular delays on the production line.

Inventory demand curves. At the time a schedule is created, a demand curve for each product is available from a corporate marketing and forecasting group. As shown in Fig. 1, each curve starts at the left with the current inventory of that product. The inventory decreases over time as future product shipments are made and eventually goes below zero if no new production occurs. To avoid penalties, the scheduler should call for more production before the demand curve falls below zero. These curves may also change over time as new information about future product demand becomes available.

Schedule costs. Running a schedule generates a dollar measure of net profit or loss. This includes the costs of running the factory, paying the workers, purchasing the raw materials, and carrying inventory at the warehouse, which are all real dollar costs. It also includes heuristic costs such as an estimate of the damage done by failing to fill a customer request when the warehouse inventory goes to zero. Finally, it includes the revenue generated from selling product to a customer. The final cost (or profit) of a schedule is the sum of all these real dollar costs, heuristic penalties, and revenue.

Given this problem description, the task of production scheduling is to maximize expected profit by selecting factory configurations over a period of time. In cases where the production rates and demand curves are assumed deterministic, the problem reduces to finding the optimal *open-loop* schedule: that is, find a fixed sequence of configurations that maximizes profit. In

the general stochastic case, the optimal choice of configuration at time t will depend on the outcomes of earlier configurations, so the optimal solution has the form of a *closed-loop* scheduling policy.

2.2 A Real Production Scheduling Problem

We have devoted considerable effort to optimizing the production scheduling of a particular U.S. factory. The physical layout of one production line in the factory is shown in Fig. 2. Raw materials enter the factory and are processed using a (proprietary) system that creates up to twelve output streams of finished products simultaneously. Depending on how numerous machines and links between machines are configured, the rate of production of each of the twelve kinds of products varies. Production costs (caused by fuel uses, personnel costs, and wasted material) also vary according to the factory configuration.

Taking into account all the constraints between machines in the factory, there are about 100,000 different possible configurations. Factories of this type typically produce on the order of \$50 million to \$2 billion worth of product annually, so the opportunities for cost savings via improved scheduling are large.

2.3 Conventional Solution Methods

Production scheduling is difficult to model within the standard job-shop scheduling paradigm. In job-shop scheduling, the problem is to complete a batch of atomic jobs under ordering constraints and constraints on which machines can handle which jobs, and at what speeds and costs. This model cannot readily be adapted to handle production rate interdependencies among machines, the desire to keep inventory levels above zero at all times (rather than just completing jobs by their deadlines), and stochasticity of demand forecasts and production.

Constraint propagation methods (e.g. [Zweben and Fox, 1994]) are commonly used to solve industrial problems. They operate by efficiently managing constraints on production deadlines and machine capabilities. Solution methods tend to search by iteratively fixing violated constraints, applying heuristics to guide the fixes. Constraint propagation focuses primarily on generating *feasible* schedules, and only secondarily on *cost optimality*. This is appropriate when feasibility is difficult, but not as good in “make to stock” scenarios where feasibility is easy and cost reduction is the main goal. Constraint propagation will not receive further consideration here for that reason.

When cost optimality is the primary scheduling objective, global optimization techniques such as simulated annealing (SA) are a good option. These methods search a space of fully-instantiated schedules to find the best ones. However, neither constraint propagation nor simulated annealing is naturally formulated to handle stochastic problems. They can be modified for nondeterminism in two ways:

- **Optimization open-loop:** Search for the fixed schedule s which maximizes the *average* profit over several independent stochastic simulations of s . Here, all the computation is spent at the beginning, and the resulting best schedule is executed without observing actual production statistics along the way. This algorithm suffers because it cannot update the schedule to account for variances in actual production. To compensate for this inadequacy, “replanning” methods are usually adopted.
- **Replanning closed-loop:** When possible, this method starts with the open-loop stochastic evaluation from the previous option. For feasibility-based methods it must start with a deterministic version of the problem. In either case, it uses its first schedule only to make some initial scheduling decisions. Then, whenever the result of an action with a stochastic outcome is observed, it replans the remainder of the schedule in order to make new decisions.

The closed-loop method can produce good results. However, it is computationally quite expensive. Moreover, although it replans on every step, its policy does not take advantage of the fact that it will be able to replan in the future—and as we show in Section 3.2 below, this dooms it to being unable to attain the optimal profit, no matter how much computation time it is allowed.

3 Production Scheduling with Value Functions

This section describes a principled approach to generating closed-loop production scheduling policies with reinforcement learning methods. The approach is based on representing the problem as an MDP and representing the solution as an approximate value function.

3.1 Production Scheduling as an MDP

Abstractly, a Markov Decision Process (MDP) is defined by a state space X , action set A , immediate reward function $R(x, a)$, and probabilistic transition model $P(x'|x, a)$. The solution to the MDP is a policy $\pi^* : X \rightarrow A$ which, if followed by the agent, will maximize the expected long-term sum of rewards attainable starting from any state x . Dynamic programming methods tabulate this optimal cumulative reward in the *optimal value function* $V^*(x)$, which is the unique solution to the *Bellman equations* [Bellman, 1957]:

$$V^*(x) = \max_{a \in A} \left(R(x, a) + \sum_{x' \in X} P(x'|x, a) V^*(x') \right) \quad (1)$$

Once V^* is computed, the optimal policy π^* is immediately obtained by choosing any action which instantiates the max in Eq. 1.

The production scheduling problem is modeled very naturally as a Markov Decision Process, as follows:

- The system state is defined by the current time $t \in 0 \dots T$; the current inventory of each product $p_1 \dots p_N$; and, if there are configuration-dependent changeover times, the current factory configuration.
- The action set consists of all legal factory configurations. We assume a discrete-time model, so the configuration chosen at time t will run unchanged until time $t + 1$.
- The stochastic transition function applies a simulation of the factory to compute the change in all inventory levels realized by running configuration c_t for one timestep. This model handles random variations in production rates straightforwardly; it also handles changeover times by simply decreasing production in proportion to the (possibly stochastic) downtime. The time t is incremented on each step, and the process terminates when $t = T$.
- The immediate reward function is computed from the inventory levels, based on the demand curve at time t . It incorporates the revenues from production, penalties from late production, employee costs, operating costs, raw material costs, and changeover cost incurred during the period. On the final time period (transition from $t = T - 1$ to T), a terminal "reward" assigns additional penalties for any outstanding unsatisfied demands.

The MDP representation suits this problem very well, for two main reasons. First, in contrast to other trajectory optimization tasks (e.g., the Travelling Salesman Problem), the utility of future decisions does not depend on the entire sequence of previous action choices and outcomes, but only on a relatively compact state description—the current time and inventory levels. Simulated annealing and other global optimization methods do not require this Markov property—nor can they exploit it. Second, the model fully represents uncertainty in production rates and changeover times. As defined here, the model also handles noise in the demands if that noise is time-independent, but it cannot account for the possibility of the demand curves being randomly updated in the middle of a schedule, since that would make the MDP transition probabilities nonstationary.

The value function for this MDP specifies a closed-loop scheduling policy which makes optimal decisions with full foresight of the remaining uncertainty in the process. No method based on global optimization can make this claim, even if replanning is allowed, as we now illustrate.

3.2 Illustrative Example

This example illustrates how MDP solutions optimally solve sequential decision problems that methods based on replanning cannot. Suppose we are asked to schedule the production of 12 units of a single product over two days. On each day we can choose one of the following three configurations:

Configuration	with probability	gets production	Cost
1	0.5 0.5	3 6	\$1
2	1.0	6	\$4
3	1.0	9	\$8

In addition to the per-configuration costs listed in the table, there is an additional cost of \$8 for each unit under 12 not produced at the end of two days. The following table shows the expected cost of each of the possible schedules. (Note that in this example, the expected cost of a schedule $[ab]$ is the same when the sequence is reversed, $[ba]$, so redundant schedules are omitted from the table.)

Config Sequence	Config Cost	Expected Missed Production Cost	Total Cost
1 1	\$2	$0.25*\$48 + 0.5*\24	\$26
1 2	\$5	$0.5*\$24$	\$17
1 3	\$9	\$0	\$9
2 2	\$8	\$0	\$8
2 3	\$12	\$0	\$12
3 3	\$16	\$0	\$16

Based on these costs, a replanning-based scheduler will choose sequence [2 2]. It will execute configuration 2 on the first day, and then have an opportunity to replan for day 2 based on the results of day 1. Since the production from configuration 2 on day 1 is deterministic (6 units), the scheduler will again choose configuration 2 on day 2, thereby completing the 2-day production run with a total cost of \$8.

The replanning-based scheduler makes a suboptimal decision on day 1 because it doesn't "know" that it will be given the chance to replan after the first day's production is observed. By contrast, with the ability to exploit this knowledge, the MDP solution makes the correct scheduling decision of action 1 on day 1. The following table evaluates the choices for day 1 by showing all the possible outcomes followed by the optimal day 2 choice for each outcome.

day 1 config	with prob	units made	day 2 config	with prob	units made	expected cost
1	0.5	3	3	1.0	9	$.5*9 + .5*5$
	0.5	6	2	1.0	6	$= \$7$
2	1.0	6	2	1.0	6	$= \$8$
3	1.0	9	1	0.5	3	$.5*9 + .5*9$
				0.5	6	$= \$9$

By considering all the possible outcomes and the optimal decisions that will be made for each one, the MDP solution chooses configuration 1 on the first day and achieves an expected cost of 7 as compared to 8 obtained by replanning. This type of tradeoff exists in real factories as well. There is often a choice of how fast to run the production line that trades off higher production rates against higher unit costs.

3.3 Value Function Approximation

In practical scheduling problems, tabulating $V^*(x)$ for every possible state of the factory is completely intractable. Instead, we use reinforcement learning methods to represent V^* compactly with a function approximator, such as global or local polynomial regression. The two methods we tested are Memory-based RTDP and ROUT.

3.3.1 Memory-Based RTDP

Memory-based RTDP is a reinforcement learning approach that is closely related to RTDP (Real-Time Dynamic Programming) [Barto *et al.*, 1995] and to Tesauro's application of TD(0) to the game of backgammon [Sutton, 1988, Tesauro, 1992]. It is also similar to the instance-based approach to representing value functions used in [Peng, 1993]. Trajectories through the MDP model are generated repeatedly, using the current approximation of the value function to guide standard Boltzmann-style exploration [Barto *et al.*, 1995]. At each step of each trajectory, a one-step backup operation (Eq. 1) is performed and the function approximator is updated.

In Memory-based RTDP, the value function is represented by a nonparametric memory-based function approximator [Cleveland and Delvin, 1988, Moore *et al.*, 1995, Atkeson *et al.*, 1995]. Memory-based learning simply accumulates training data points, rather than running a training algorithm on them. Then whenever a query is made, the approximator's output is computed by a weighted average or weighted polynomial regression over nearby points in memory.

Achieving good performance with Memory-based RTDP requires an appropriate choice of the Boltzmann exploration temperature and the local regression kernel width. These values were tuned empirically to obtain the results presented in Section 4. Although the training points generated by Memory-based RTDP's early trajectories are undoubtedly inaccurate samples of V^* , we did not find it necessary to include an explicit "forgetting" mechanism in the learning; the bad points are quickly overwhelmed by later, more accurate samples.

3.3.2 ROUT

ROUT is an active learning algorithm for value function approximation that is specifically designed for the subclass of acyclic MDPs [Boyan and Moore, 1996]. Note that the scheduling MDP is certainly acyclic, since its state representation includes the time counter t . Using simulations of the process, ROUT repeatedly identifies a new state x at which (1) the function approximator is currently in error, and (2) an accurate sample of V^* can be obtained from a 1-step backup. Unlike Memory-based RTDP and most other reinforcement learning methods, ROUT explicitly tries to prevent the function approximator from seeing *any* inaccurate samples of V^* .

Details of how ROUT identifies such states automat-

ically are given in [Boyan and Moore, 1996]. One by one, these useful states are accumulated into a training set of accurate samples of $V^*(x)$. The training set grows backwards from the terminal states. As soon as the start state x_0 is itself added to the training set, ROUT declares victory, outputs its learned training set and learned approximation of V^* , and terminates.

If the function approximator cannot represent V^* accurately, then ROUT may become stuck, repeatedly adding points near the terminal states and never progressing backwards. However, if the function approximator can represent V^* to within the specified tolerance, then ROUT can be guaranteed to eventually find it. For ROUT to find V^* efficiently, the function approximator must extrapolate well from a small training set.

4 Experimental Results

We have experimented with two instances of the real-world production scheduling task described in Section 2.2. The first instance is heavily simplified so that the exact optimal closed-loop scheduling policy can be calculated tractably. The second instance is a more realistic model, for which only heuristic solutions are available.

4.1 Simplified Scheduling Instance

In the simplified instance, the task is to schedule 8 weeks of production; however, configurations may be changed only at 2-week intervals, and only 17 configuration choices are available. Of these 17, nine have deterministic production rates; the other eight each have two stochastic outcomes, producing only 1/3 of their usual amount with probability 0.5. With a total of $9 \times 1 + 8 \times 2 = 25$ outcomes possible from every state, there are $25^4 = 390,625$ possible trajectories through the space. The optimal policy can be computed by tabulating $V^*(x)$ at every possible intermediate state x of the factory, of which there are $1 + 25 + 25^2 + 25^3 = 16,276$. The optimal policy results in an expected cumulative reward of $-\$22.8M$. By contrast, a random schedule attains a reward of $-\$923M$ on average! A greedy policy, which at each step selects a configuration to maximize only the one-step reward from the current state, attains $-\$97.9M$.

We applied ROUT to this instance, trying three different function approximators: 1-nearest neighbor, locally-weighted linear regression, and global quadratic regression. KD-trees were used to keep the

computation efficient [Moore *et al.*, 1997]. For the locally weighted regression, a kernel width of 2^{-3} of the range of each input dimension in the training data was used. ROUT's exploration and tolerance parameters were tuned manually. Table 1 summarizes the results.

When nearest-neighbor was used as the function approximator, ROUT did not obtain sufficient generalization from its training set and failed to terminate within a limit of several hours. However, with both local linear and global quadratic regression models, ROUT did run to completion and produced an approximate value function which significantly outperformed the greedy policy. Moreover, over half of the ROUT runs did indeed terminate with the *optimal* closed-loop scheduling policy. In these cases, ROUT's final self-built training set for value function approximation consisted of only about 100–150 training points—a substantial reduction over the 16,276 required for full tabulation of V^* . ROUT's total running time (≈ 1 hour on a 200 MHz Pentium Pro) was roughly half of that required to enumerate V^* manually.

From these preliminary results, we conclude that ROUT does indeed have the potential to approximate V^* extremely well, given a suitable function approximator for the domain. However, since it runs quite slowly on even this simplified problem, we believe ROUT will not scale up to practical scheduling instances without further refinements.

4.2 Practical Scheduling Instance

In this section we present experimental results on a larger scheduling problem. In doing so, we lose the ability to determine the optimal policy for comparison. However, it gives a better demonstration of how the competing methods perform on industrial-scale scheduling problems. The task is to schedule eight weeks of production at one week intervals. There are eight products, eight machines, and a total of 421 legal configurations to consider, including the "closed" configuration.

Our experiments consider both deterministic and noisy versions of the problem. To build the deterministic version of the problem, we ran long (stochastic) simulations for each of the 421 actions and cached the mean observed production rate for each. For the noisy versions, we could have used the noisy outcomes directly from the stochastic simulation, but instead we simply added Gaussian noise to the cached, deterministic production rates. This enabled our experiments to run significantly faster, and also allowed us to eas-

Algorithm	Mean Profit	95% C.I.	optimal runs
Optimal	-22.8		1
Random	-923.2	± 58.7	0
Greedy	-97.9	± 15.1	0
ROUT + global quadratic	-57.0	± 23.5	10/16
ROUT + local linear	-45.0	± 16.9	10/16

Table 1: Results for 4-timestep, 17-configuration stochastic scheduling problem.

ily generate empirical results with varying amounts of noise.

Table 2 shows experimental results. The computation times reported are on a 200 MHz Pentium Pro. The first section contains results for the case where the factory output is deterministic and known. The purpose of the first two lines is to delimit the range of results we should expect from good algorithms. The "Random" algorithm builds a schedule by choosing 8 configurations at random, and it loses an enormous amount of money. Much of the cost is due to heuristic penalties for failing to satisfy customer demand.

The "PlanIt" algorithm, developed by Schenley Park Research, is the proprietary algorithm currently used to schedule the real factory's production. It has several advantages over the other algorithms in this table. First, it is finely tuned to schedule this factory using a combination of simulated annealing, linear programming, constraint propagation, and several heuristics. Second, it is not restricted to choosing configurations for pre-discretized time steps, but can choose an arbitrary number of configurations and switch between them at arbitrary times. Our experience with this scheduler leads us to believe that the average profit of \$13.81M is very near optimal for this instance, so it can be considered an unattainable upper bound for the other results. In particular, PlanIt achieves its results by using an average of around 13 configurations in its schedules while the other algorithms are restricted to 8 fixed-sized time steps. It usually incurs no heuristic penalties in its schedules, so that figure is a profit in real dollars.

The simulated-annealing, greedy-exploration, and Memory-based RTDP algorithms are run as described in the previous sections. The simulated annealing runs made use of the successful "modified Lam" adaptive annealing schedule [Ochotta, 1994]. Memory-based RTDP used kernel regression with a kernel width of 2^{-5} of the range of each state variable, and used KD-trees for efficiency [Moore *et al.*, 1997]. Boltzmann exploration (without cooling) was used for the deter-

ministic case, but proved unnecessary in the stochastic case because the noise alone caused sufficient exploration.

The poor result from Greedy in the deterministic case shows that generating trajectories based solely on the one-step cost of configurations is not an effective way to search, even when compared to a randomized search method such as simulated annealing. The search efficiency gained by computing a value function is shown by the favorable Memory-based RTDP results. They are obtained from only 200 trajectories through the state space, meaning the value function at each time step is represented with 200 training points. All of the algorithms can do better with more computation time, but they were cut off at 10 minutes since PlanIt gets its results in that much time.

The second and third sections of the table show results with 10% and 20% noise added. The PlanIt algorithm cannot be run in these cases since it does not handle stochastic outcomes; however, we still expect its result in the deterministic case to be a reasonable upper bound for the other algorithms.

Open-loop simulated annealing means that all the computation is spent at the beginning and the resulting best schedule is executed without observing actual production statistics along the way. This algorithm suffers because it cannot update the schedule to account for variances in actual production. By contrast, closed-loop simulated annealing replans the rest of the schedule after each week of actual production is observed. In order to keep the total computation the same, the computation allotted for each week's decision was divided by the number of weeks (8). The results show that replanning does improve over open loop execution. We note that all the simulated-annealing schedulers have high variance, which can be a disadvantage of using that algorithm.

Memory-based RTDP uses its computation at the beginning to compute a value function. Each run used 400 trajectories for these results. The value function determines a closed-loop policy valid for any state

Noise level	Algorithm	Mean Profit	95% C.I.
Deterministic (≈ 10 min computation)	Random	-466.35	± 59.45
	PlanIt	13.81	± 0.08
	Simulated Annealing	5.66	± 3.68
	Greedy + Exploration	-1.93	± 3.21
	Memory-based RTDP	7.70	± 1.57
10% Noise (≈ 45 min computation)	Greedy (c.l.)	-17.69	± 1.94
	Simulated Annealing (o.l.)	6.48	± 1.21
	Simulated Annealing (c.l.)	9.03	± 1.04
	Memory-based RTDP	10.16	± 0.84
20% Noise (≈ 45 min computation)	Greedy (c.l.)	-25.92	± 1.12
	Simulated Annealing (o.l.)	2.55	± 1.91
	Simulated Annealing (c.l.)	2.40	± 3.95
	Memory-based RTDP	7.02	± 0.67

Table 2: Results for 8-timestep, 421-configuration scheduling problem. The numbers shown represent profits in millions of dollars. On the noisy problems, Memory-based RTDP is statistically better than the other algorithms at the 95% significance level.

reached during actual production. As discussed earlier, it not only executes closed-loop, but also makes its decisions "knowing" that it will be executing closed-loop. The results show both a favorable expected profit as well as smaller variance across runs.

5 Discussion and Future Work

We expect Memory-based RTDP to outperform simulated annealing on a stochastic problem based on the intuition from Sec. 3.2, and our experimental results show that it does. It is interesting to observe that Memory-based RTDP does well against simulated annealing even in the deterministic case where the stochastic modeling capability of MDPs is not needed. This provides further evidence that search based on value functions can improve efficiency. While simulated annealing is forced to try configurations at random, value function based methods can explicitly reason about which intermediate states are good and which actions will reach those states.

To our knowledge, this work represents the first application of reinforcement learning to production scheduling with multiple products made on multiple machines. The scheduling of machine maintenance is discussed in [Mahadevan *et al.*, 1997], and transfer line production scheduling is discussed in [Mahadevan and Theocharous, 1998]. In their task, each product or sub-product is produced on a single machine and each machine makes a local decision on whether to produce one of its products or go down for maintenance. A

reinforcement learning approach to the Space Shuttle scheduling problem is described by [Zhang and Dietterich, 1995]. In that framework, states are complete schedules and actions are modification operators applied to the schedules. Their feature representation introduces noise, but the underlying problem is deterministic.

Our empirical work to date covers stochasticity only in production. Another large source of uncertainty in real problems is the inadequacy of demand forecasts. This can be handled heuristically within the MDP formulation described here by the addition of appropriate noise to the demands during simulations. However, it may also be possible to gain extra efficiencies by incorporating demands explicitly into the MDP state space. Further empirical work is required to answer that question.

As the size of the scheduling problem increases, it becomes increasingly expensive to compute the value function accurately. However, even an inexact value function can be useful as the basis for a quasi-greedy search or "rollout" search performed online [Tesauro and Galperin, 1997]. We intend to test such methods in future work on larger scheduling problems.

Acknowledgements

The second author acknowledges the support of a NASA GSRP fellowship. The third author acknowledges the support of an NSF Career Award.

References

- [Atkeson *et al.*, 1995] C. Atkeson, S. Schaal, and A. Moore. Locally weighted learning. *AI Review*, 1995.
- [Barto *et al.*, 1995] A. G. Barto, S. J. Bradtke, and S. P. Singh. Real-time learning and control using asynchronous dynamic programming. *Artificial Intelligence*, 1995.
- [Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [Boyan and Moore, 1996] J. A. Boyan and A. W. Moore. Learning evaluation functions for large acyclic domains. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*. Morgan Kaufmann, 1996.
- [Cleveland and Delvin, 1988] W. Cleveland and S. Delvin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, pages 596–610, September 1988.
- [Mahadevan and Theocharous, 1998] S. Mahadevan and G. Theocharous. Optimizing production manufacturing using reinforcement learning. In *Eleventh International FLAIRS Conference*, 1998.
- [Mahadevan *et al.*, 1997] S. Mahadevan, N. Marchal-leck, T. Das, and A. Gosavi. Self-Improving Factory Simulation using Continuous-Time Average-Reward Reinforcement Learning. In *Proceedings of the 14th International Conference on Machine Learning (IMLC '97)*, Nashville, TN. Morgan Kaufmann, July 1997.
- [Moore *et al.*, 1995] A. Moore, C. Atkeson, and S. Schaal. Locally weighted learning for control. *AI Review*, 1995.
- [Moore *et al.*, 1997] A. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In *International Conference on Machine Learning*, 1997.
- [Ochotta, 1994] E. Ochotta. *Synthesis of High-Performance Analog Cells in ASTRX/OBLX*. PhD thesis, Carnegie Mellon University Department of Electrical and Computer Engineering, April 1994.
- [Peng, 1993] J. Peng. Efficient Dynamic Programming based Learning for Control. PhD. Thesis, Northeastern University, December 1993.
- [Sutton, 1988] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.
- [Tesauro and Galperin, 1997] G. Tesauro and G. R. Galperin. On-line policy improvement using Monte-Carlo search. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.
- [Tesauro, 1992] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4), May 1992.
- [Zhang and Dietterich, 1995] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1114–1120, 1995.
- [Zweben and Fox, 1994] M. Zweben and M. Fox. *Intelligent Scheduling*. Morgan Kaufmann, 1994.

Heading in the Right Direction

Hagit Shatkay

Leslie P. Kaelbling

Department of Computer Science

Brown University

Providence, RI 02912

{hs, lpk}@cs.brown.edu

Abstract

Stochastic topological models, and hidden Markov models in particular, are a useful tool for robotic navigation and planning. In previous work we have shown how weak odometric data can be used to improve learning topological models, overcoming the common problems of the standard Baum-Welch algorithm. Odometric data typically contain directional information, which imposes two difficulties: First, the cyclicity of the data requires the use of special circular distributions. Second, small errors in the heading of the robot result in large displacements in the odometric readings it maintains. The cumulative rotational error leads to unreliable odometric readings. In the paper we present solutions to these problems by using a circular distribution and relative coordinate systems. We validate their effectiveness through experimental results from a model-learning application.

1 INTRODUCTION

Directional data is information consisting of magnitude and *direction*. Such data is an integral part of important applications in various areas of computer science in general and artificial intelligence in particular. In computer graphics, automatic production of pen-and-ink drawings and the production of animation based on magnetic trackers data requires statistical manipulation of directional data. In cognitive science, modeling routes chosen by animals [4] requires a similar kind of statistical manipulation. In the area of machine learning we often use probabilistic models for robot movement. Most aspects of robot movement (arm movement as well as the whole body movement) can be described in terms of location and heading change, requiring the use and manipulation of directional data.

Probabilistic models are widely used within the AI community. Such models may allow continuous probabilities, as demonstrated in work on Bayesian networks [7], hidden Markov models [5, 8], probabilistic clusters [2] and stochastic maps [19], to name a few. However, the assumption underlying all the above work is that continuous distributions are *linear* — that is — distributions that assign density to each point on the real line so that the area under the density curve, integrated over the whole real line, is 1.¹ Such models do not take into account directional data, which is inherently *cyclic*. Under circular distributions the density of any point x on the real line is the same as that of $x + k\xi$ where k is any integer and ξ is some real number.

The need for circular distributions has long been realized by statisticians [6], but the practice of using them has not found its way into the computer science community and to the machine learning community in particular. One of the goals of this paper is to point out the usefulness of one specific circular distribution in the context of robotics, and provide a short tutorial on circular distributions.

Another special aspect of directional data is its sensitivity to errors. As most navigators, pilots and skippers have experienced, a small angular deviation from the original course causes a big displacement at the final location. This problem is very prominent in mobile robots, where drifts and drags of the wheels and disalignment of both engines and floors can cause a robot to face in the wrong heading with respect to its own odometric readings. Odometric information is recorded by the robot along three dimensions; it consists of the changes along the x and the y axis as well as a change in the *heading* of the robot within a global coordinate system. In our previous work on learning topological models [17] we made several assumptions about the odometric data:

- All odometric measures are normally distributed.

¹Most often the distribution is Gaussian.

- All corridors are perpendicular to each other.
- The robot, when collecting the data, is using the perpendicularity assumption, and is collecting the data with respect to one global coordinate system.

This paper demonstrates the problematic aspects of these assumptions and introduces our solution to the problems, together with preliminary results that demonstrate the effectiveness of our solution. The rest of the paper is organized as follows: Section 2 describes our application and motivates the need for circular distributions in the context of machine learning; Section 3 presents the von Mises distribution, which is a circular version of the normal distribution; Section 4 discusses the problems faced due to heading deviations and presents our solution to the problem; Section 5 presents experiments and results to demonstrate the usefulness of our approach; Section 6 concludes the paper.

2 LEARNING TOPOLOGICAL MODELS

Hidden Markov models (HMMs), as well as their generalization to models for partially observable Markov decision processes (POMDP models), are a useful tool for representing environments such as road networks and office buildings, which are typical for robot navigation and planning [1, 14, 18]. Previous work on planning with such models typically assumed that the model is manually provided. Manual acquisition of these models can be very tedious and hard. It is desirable to learn such models automatically, both for robustness and in order to cope with new and changing environments. Since POMDP models are a simple extension of HMMs, they can, theoretically, be learned with a simple extension to the Baum-Welch algorithm [15] for learning HMMs. However, without a strong prior constraint on the structure of the model, the Baum-Welch algorithm does not perform very well: it is slow to converge, requires a great deal of data, and often becomes stuck in local maxima. In previous work [16, 17] we demonstrated how the simple Baum-Welch algorithm can be enhanced with weak local odometric information to learn better models faster, under the assumption listed above. For the sake of completeness, we briefly review the essentials of this work here.

A robot moves through the corridors in an office environment. Low-level software provides a level of abstraction that allows the robot to move through hallways from intersection to intersection and turn ninety degrees to the left or right. At each intersection, ultrasonic data interpretation lets the robot observe, in each of the four cardinal directions, whether there is an open space, a door, a wall, or something unknown. The robot also has encoders on its wheels that allow it to estimate its current pose (position and orientation) with respect to its pose at the previous intersection. Of course, the action and perception routines

and the odometric measures are all subject to error. The learning task is to deduce a model from the recorded observations and odometric information.

Our learning algorithm gets as an input an experience sequence E of observations and odometric readings, and produces as output an HMM², λ , of the environment, such that the likelihood, $Pr(E|\lambda)$, is locally maximized. Formally, the standard HMM is defined as a tuple $\lambda = \langle S, O, A, B, \pi \rangle$, where:

- $S = \{s_1, \dots, s_N\}$ is a finite set of N states;
- $O = \prod_{i=1}^l O_i$ is a finite set of observation vectors length l ; the i th element of an observation vector is chosen from the finite set O_i ;
- A is a stochastic transition matrix, with $A_{i,j} = Pr(q_{t+1} = s_j | q_t = s_i)$; $1 \leq i, j \leq N$; q_t is the state at time t ;
- B is an array of l stochastic observation matrices, with $B_{i,j,o} = Pr(V_t[i] = o | q_t = s_j)$; $1 \leq i \leq l$, $1 \leq j \leq N$, $o \in O_j$; V_t is the observation vector at time t ;
- π is a stochastic initial probability vector describing the distribution of the initial state.

Odometric information gathered by the robot is not an inherent part of the topological model, but is used by the learning algorithm to better identify and distinguish states. To facilitate the use of this information we augment the standard model with the odometric relation matrix:

- R is a relation matrix, specifying for each pair of states, s_i and s_j , the mean and variance of the D -dimensional metric relation between them; $\mu_{ij}^d \stackrel{\text{def}}{=} \mu(R_{i,j}[d])$ is the mean of the d^{th} component of the relation between s_i and s_j and $(\sigma_{ij}^d)^2 \stackrel{\text{def}}{=} \sigma^2(R_{i,j}[d])$, the variance, where $1 \leq d \leq D$. Furthermore, R is *geometrically consistent*: for each component d , the relation $R^d(a, b) \stackrel{\text{def}}{=} \mu(R_{a,b}[d])$ must satisfy the following properties for all states a, b , and c :
 - ◊ $R^d(a, a) = 0$;
 - ◊ $R^d(a, b) = -R^d(b, a)$ (*anti-symmetry*); and
 - ◊ $R^d(a, c) = R^d(a, b) + R^d(b, c)$ (*additivity*);

The odometric information recorded by the robot at time t , r_t , consists of the change in the x and y coordinates of the odometric readings when moving from state q_{t-1} to state q_t , as well as the change of the robot's heading, θ , between these states.

An arbitrary initial model λ_0 is assumed. Then an expectation maximization algorithm [3] is executed as follows:

²We discuss here HMMs rather than POMDP models. Extension to POMDPs is straightforward, but notationally more cumbersome.

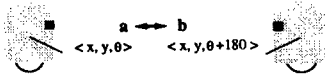


Figure 1: Robot changes heading from state a to state b .

- *E-step*: computes the state-occupation and transition probabilities, $\gamma_t(i) = Pr(q_t = s_i | E, \lambda)$ and $\xi_t(i, j) = Pr(q_t = s_i, q_{t+1} = s_j | E, \lambda)$, respectively, at each time t in the sequence, given E and the current model λ , and
- *M-step*: finds a new model λ that maximizes $Pr(E | \lambda, \gamma, \xi)$.

Introducing odometric information requires iterative updates of the odometric relations between pairs of states, in the relation matrix, R . The updates need to maintain the properties listed above, although currently the update procedure only satisfies the first two.

The learning task is further complicated by the special nature of the heading reading and the rotational errors accrued. The following section goes in more detail into the special issues of handling the heading information. The rest of the paper deals with resolving the problems caused by rotational errors.

3 DIRECTIONAL DATA AND DISTRIBUTIONS

Suppose a robot is in state a , which is in location $\langle x, y \rangle$ facing in direction θ , as shown in figure 1. By turning backwards, it transitions to state b , and a respective change of heading of approximately $\pm 180^\circ$ is recorded. Thus the new recorded configuration of the robot is $\langle x + \epsilon_1, y + \epsilon_2, \theta \pm 180^\circ + \epsilon_3 \rangle$, where ϵ_i is the error due to inaccuracy in both measurement and movement. In earlier work [17], we treated all errors — in both location (x, y) and heading (θ) — as if they were normally distributed. However, the change in heading is different from changes in x and y , since angular measurements are *cyclic*. That is, a change in heading of θ° is the same as that of $\theta \pm 360^\circ k$, for any integer k .

If we knew in advance, for every pair of states, the approximate change in heading ($\Delta\theta$) between them, we could have modeled it as normal with mean $\Delta\theta$, and small variance σ^2 . We could have adopted a convention, normalizing all angles to be within a cyclic range, e. g. $[-180^\circ, 180^\circ]$, (similarly we may use radians), and always chosen to take as the angular change between two points $\min(|\Delta\theta|, 360^\circ - |\Delta\theta|)$, and assigned it the correct sign. Such an approach of using a non-circular distribution is justified when the estimation of a position is based only on readings a-priori known to be taken near this position, (see for example work by Thrun et al [20] and Lu et al [12]).

However, we do not know in advance the angles between states. The data is a sequence of measurements recorded at all the states. We estimate the probabilities of the states in which they were recorded, and take a *weighted mean* of the measurements in order to estimate the angular change between every two states. Thus, we are facing the following problem: *What is the interpretation of a "mean angle"?*

As an example, suppose we want to estimate the heading change from state a to state b of Figure 1. We adopt the convention of angles being expressed between -180° and 180° . Also, suppose that the robot recorded two measurements of angular distance from state a to state b : -169° and 185° . The simple average between these measurements is an estimate of the mean heading change of 8° . Obviously this value does not even approximate the change of heading between the two states. The same problem arises if we use any other convention for expressing angles (e. g. 0° to 360°). The problem lies in the fact that angles that are about 180° away from the mean angle, indeed greatly deviate from this mean, while angles that deviate about 360° are actually very close to it. To capture this idea, the concept of *circular distribution* is required. We provide a brief introduction to the concepts and techniques used for handling directional data. In particular we concentrate on the *von Mises distribution* — a circular version of the normal distribution. Further discussion can be found in the statistical literature [6, 10, 13]. Section 3.3 returns to show how the theory is applied in our model and learning algorithm.

3.1 STATISTICS OF DIRECTIONAL DATA

Directional data in the 2-dimensional space can be represented as a collection of 2-dimensional vectors, $(\langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle)$, on the unit circle, as shown in Figure 2. The points can also be represented as the corresponding angles between the radii from the center of the unit circle and the x axis, $(\theta_1, \dots, \theta_n)$, respectively. The relationship between the two representations is:

$$x_i = \cos(\theta_i), \quad y_i = \sin(\theta_i), \quad (1 \leq i \leq n).$$

The vector mean of the n points, (\bar{x}, \bar{y}) , is calculated as:

$$\bar{x} = \frac{\sum_{i=1}^n \cos(\theta_i)}{n}, \quad \bar{y} = \frac{\sum_{i=1}^n \sin(\theta_i)}{n}. \quad (1)$$

Using polar coordinates, we can express the mean vector in terms of angle, $\bar{\theta}$, and length, \bar{a} , where (except for the case $\bar{x} = \bar{y} = 0$):

$$\bar{\theta} = \arctan\left(\frac{\bar{y}}{\bar{x}}\right), \quad \bar{a} = (\bar{x}^2 + \bar{y}^2)^{\frac{1}{2}} \quad (2)$$

The angle $\bar{\theta}$ is the mean angle, while the length \bar{a} is a measure (between 0 and 1) of how concentrated the sample angles are around $\bar{\theta}$. The closer \bar{a} is to 1, the more concentrated the sample is around the mean, which corresponds to a smaller sample variance.

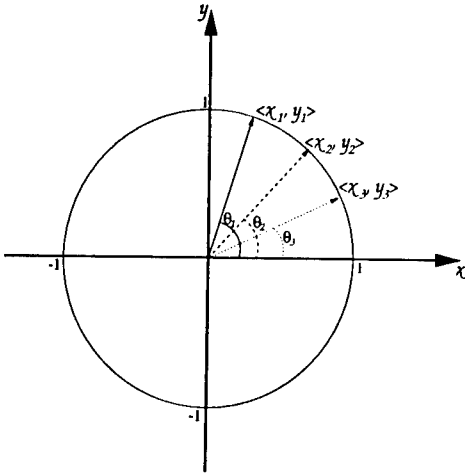


Figure 2: Directional data represented as angles and as vectors on the unit circle.

A function f is a density function of a continuous *circular distribution* if and only if: $f(x) \geq 0$ and $\int_0^{2\pi} f(x)dx = 1$. A simple example of a circular distribution is the *uniform circular distribution*, whose density function is $f(\theta) = \frac{1}{2\pi}$ (where θ is measured in radians).

One way of deriving a circular version of an unlimited linear distribution is through “wrapping” it around a circumference of the unit circle. If x is a random variable on the line with probability density function $f(x)$, the wrapped random variable $x_w = [x \bmod 2\pi]$ is distributed according to a wrapped distribution with the probability density function: $f_w(\theta) = \sum_{k=-\infty}^{\infty} f(\theta + 2\pi k)$. Applying this derivation to the normal distribution results in a circular version of the normal distribution, but estimating its parameters from sample data can be hard [6, 13]. An easier-to-estimate circular version of the normal distribution was derived, by von Mises [6, 13]. We use this distribution to model the robot heading in this work, and it is described below.

3.2 THE VON MISES DISTRIBUTION

A circular random variable, θ , $0 \leq \theta \leq 2\pi$, is said to have the *von Mises distribution* with parameters μ and k , where $0 \leq \mu \leq 2\pi$ and $k > 0$, if its probability density function is:

$$f_{\mu,k}(\theta) = \frac{1}{2\pi I_0(k)} e^{k \cos(\theta - \mu)},$$

where $I_0(k)$ is the modified Bessel function of the first kind and order 0:

$$I_0(k) = \sum_{r=0}^{\infty} \frac{1}{r!^2} \left(\frac{1}{2}k\right)^{2r}.$$

Similar to the linear normal distribution, this is a unimodal distribution, symmetrical around μ . The mode is at $\theta = \mu$ while the antimode is at $\theta = \mu + \pi$. We observe that the ratio of the density at the mode to the density at the antimode is e^{2k} , which indicates that the larger k is, the more concentrated the density is about the mode. Figure 3 shows an

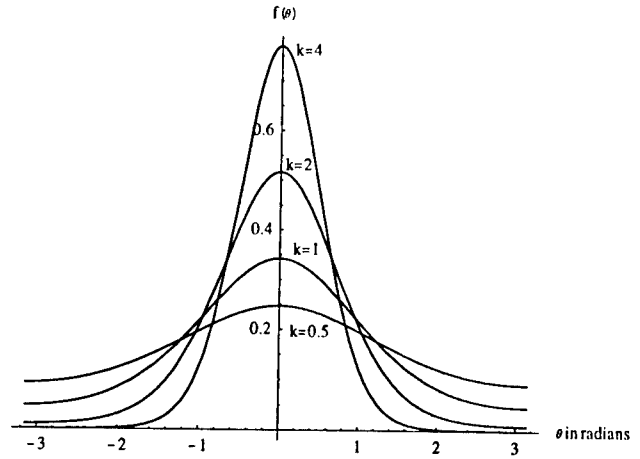


Figure 3: The von Mises distribution with mode 0 and various k values.

“unwrapped” plot of the von Mises distribution for various values of k where $\mu = 0$.

We now describe how to estimate the parameters μ and k given a set of heading samples (angles $\theta_1, \dots, \theta_n$) from a von Mises distribution [13]. We are looking for maximum likelihood estimates for μ and k . The likelihood function for the data generated by a von Mises distribution with parameters μ and k is:

$$L_{\mu,k} = \prod_{i=1}^n f_{\mu,k}(\theta_i) = \frac{e^{k \sum_{i=1}^n \cos(\theta_i - \mu)}}{(2\pi)^n I_0(k)^n}.$$

The maximum likelihood estimate for μ , $\bar{\mu}$, is: $\bar{\mu} = \arctan(\frac{\bar{y}}{\bar{x}})$, where \bar{y} , \bar{x} are as defined in equation 1.

The maximum likelihood estimate for k is the \bar{k} that solves the equation:

$$\frac{I_1(\bar{k})}{I_0(\bar{k})} = \frac{1}{n} \sum_{i=1}^n \cos(\theta_i - \mu). \quad (3)$$

If we don't know μ and are only interested in estimating k with respect to the *estimate* $\bar{\mu}$, by using trigonometric manipulation and the definition of $\bar{\mu}$ (Equation 2), we can substitute the right hand side of equation 3 by $\bar{\mu}$ and obtain that the maximum likelihood estimate for k is \bar{k} that satisfies: $\frac{I_1(\bar{k})}{I_0(\bar{k})} = \bar{\mu}$.

However, if we do have a given μ and want to find a maximum likelihood estimate for the concentration k of the sample data around that specified μ , we need to use as a maximum likelihood estimate for k , \bar{k} that satisfies:

$$\frac{I_1(\bar{k})}{I_0(\bar{k})} = \frac{1}{n} \sqrt{\left(\sum_{i=1}^n \cos(\theta_i)\right)^2 + \left(\sum_{i=1}^n \sin(\theta_i)\right)^2 - \left(\sum_{i=1}^n \sin(\mu - \theta_i)\right)^2}.$$

The above estimation formulae agree with the intuition that the sample is more concentrated (\bar{k} is larger) about the sample mean ($\bar{\mu}$) than about the true distribution mean (μ).

The rest of the section explains how the von Mises parameters are incorporated into the Hidden Markov model, and how the learning algorithm is adapted to learn these parameters.

3.3 HANDLING ANGULAR ODOMETRIC READINGS

To model the heading difference between each pair of states, the relation matrix R , described in Section 2, is 3-dimensional, consisting of the components $\langle x, y, \theta \rangle$. The component $R_{i,j}[\theta]$ represents the heading change of moving from state s_i to s_j , and is assumed to be distributed according to the von Mises distribution. The notation $\mu_{i,j}^\theta \stackrel{\text{def}}{=} \mu(R_{i,j}[\theta])$ represents the mean of the distribution for this heading change, while $k_{i,j}^\theta \stackrel{\text{def}}{=} k(R_{i,j}[\theta])$ represents the concentration parameter around the mean³. The three constraints described before for the components of R , (ideally) hold for the θ component as well.

Similarly, every observed relation item, r_t , in the experience sequence E , has a heading-change component, θ , which records the robot's estimated change in heading between the state at time t , q_t , and the state q_{t+1} .

The reestimation formula for the von Mises mean parameter of the heading change between states s_i and s_j is:

$$\bar{\mu}_{i,j}^\theta = \arctan \left(\frac{\sum_{t=0}^{T-2} [\sin(r_t[\theta])\xi_t(i,j) - \sin(r_t[\theta])\xi_t(j,i)]}{\sum_{t=0}^{T-2} [\cos(r_t[\theta])\xi_t(i,j) + \cos(r_t[\theta])\xi_t(j,i)]} \right).$$

The fraction denotes the ratio between the expected sine and the expected cosine of the heading change from state i to state j . Since the heading change from j to i is identical in magnitude but opposite in direction to the heading change from i to j , the transitions from j to i are also accumulated – with reversed signs. By taking \arctan of this ratio we get an estimate for the mean heading change itself.

To reestimate the concentration parameter, we need to find $\bar{k}_{i,j}^\theta$ such that:

$$\frac{I_1[\bar{k}_{i,j}^\theta]}{I_0[\bar{k}_{i,j}^\theta]} = \frac{\sum_{t=0}^{T-2} [\xi_t(i,j) \cos(r_t[\theta] - \bar{\mu}_{i,j}^\theta)]}{\sum_{t=0}^{T-2} \xi_t(i,j)}.$$

³In contrast, x and y are normally distributed and have their variance rather than concentration stored in R .

Finding $\bar{k}_{i,j}^\theta$ that satisfies this equation is done through the use of a lookup table listing values of the quotient $\frac{I_1[x]}{I_0[x]}$.

The above reestimation formulae agree with the maximum likelihood estimator formulae given in Section 3.1. Their correctness can be proved along the lines of the proof provided in our previous document [16].

4 STATE-RELATIVE COORDINATE SYSTEMS

In our previous work we assumed that there is a single global coordinate system within which the robot operates. Moreover, we assumed that the robot collects its data within a perpendicular corridor framework and that it takes advantage of this single perpendicular framework while recording odometric information. This assumption may be troublesome in practice. The rest of the paper discusses the potential problems, presents a method for relaxing the assumptions and addressing the problems, and demonstrates the effectiveness of the solutions through experiments and results.

4.1 MOTIVATION

We tend to think about an environment as consisting of landmarks fixed in a global coordinate system and corridors or transitions connecting these landmarks. However, this view may be problematic when robots are involved.

Conceptually, a robot has two levels in which it operates; the *abstract* level, in which it centers itself through corridors, follows walls and avoids obstacles, and the *physical* level in which motors turn the wheels as the robot moves. In the physical level many inaccuracies can occur: unaligned wheels or unsynchronized motors can cause sideways drift, an obstacle under a wheel can cause the robot to slightly rotate around itself, or uneven floors may cause the robot to slip in a certain direction. In addition, the odometric measuring instrumentation may be inaccurate in and of itself. In the abstract level, corrective actions are constantly executed to overcome the physical drift and drag. For example, if the left wheel is disaligned and drags the robot leftwards, a corrective action of moving to the right is constantly taken in the higher level to keep the robot centered in the corridor.

Such phenomena greatly effect the odometry recorded by the robot, if it is interpreted with respect to one global framework. For example, consider the robot depicted in Figure 4. It drifts to the left $-\phi^\circ$ when moving from one state to the next, and corrects for it by moving ϕ° to the right to maintain itself centered in the corridor, moving along the solid arrow. Let us assume that states are lo-

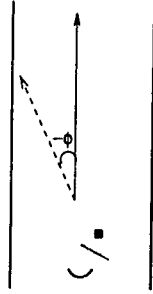


Figure 4: The robot moves in a corridor along the solid arrow, correcting for drift in the direction of the dashed arrow.

cated along the center of the corridor, which is aligned with the y axis of the global coordinate system. The robot steps back and forth in the corridor. Whenever it reaches a state, its odometry reading changes by $\langle x, y, \theta \rangle$ along the $\langle X, Y, \text{heading} \rangle$ dimensions, respectively. As the robot proceeds, the deviation with respect to the x axis becomes more and more severe. Thus, after going through several transitions, the odometric changes recorded between every pair of states, with respect to a global coordinate system, become larger and larger (especially in the X dimension).

Similar problems of inconsistent odometric changes recorded between pairs of states can arise along any of the odometric dimensions. It is especially severe when such inconsistencies arise with respect to the heading, since this can lead to confusion between the X and the Y axes, as well as confusion between forwards and backwards movement (when the deviation in the heading is around 90° or 180° respectively). An example of our robot view of a perfectly perpendicular office environment, based on its odometric readings within a global coordinate system, is shown in Figure 5. The data was collected by our robot Ramona, while moving along the corridors in an area of our department, depicted in Figure 7.

A solution to such a situation is to model the odometric relations of moving from state s_i to state s_j using a changing coordinate system which is *respective* to state s_i , as opposed to a global coordinate system anchored at the initial state. We formalize this idea and provide the update rules for the odometric information based on this approach in the rest of this section. We have implemented our solution, and demonstrate its effectiveness throughout Section 5.

4.2 LEARNING ODOMETRIC RELATIONS WITH CHANGING COORDINATES

As before, our experience sequence E consists of T pairs $\langle r_t, V_t \rangle$ of recorded odometric relations and observation vectors. The odometric relations are still recorded with respect to the robot's global coordinate system. However, when learning the relation matrix from the odometric readings, we interpret the entry $R_{i,j}$ in the relation matrix R , as encoding the information with respect to a coordinate sys-

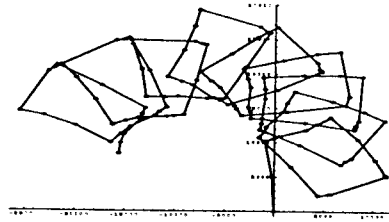


Figure 5: A path in a perpendicular environment, plotted based on odometric readings taken by the robot Ramona.

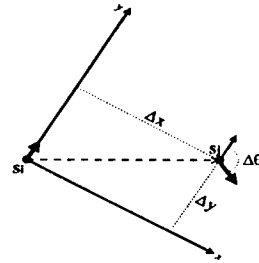


Figure 6: Robot in state s_i , facing in the direction of the y axis.

tem whose origin is anchored at the state s_i ; the y axis is aligned with the robot's heading in state s_i and the x axis is perpendicular to it. This is depicted in figure 6. The robot is in state s_i facing in the direction pointed to by the y axis. Its relationship to the state s_j is described in terms of the coordinate system shown in the figure. Its heading in each state is denoted by the bold arrow.

To support this interpretation of the relation matrix we need to revisit the formulation of the geometrical-consistency constraints stated in Section 2, as well as the update formulae used when learning the model.

The consistency constraints have to reflect the coordinate system with respect to which the odometry is represented. Since the heading measurement is independent of any specific coordinate system, only the constraints over the x and y components of the odometric relation need to be redefined. We denote by $\mu^{(x,y)}(a,b)$ the vector $\langle \mu(R_{a,b}[x]), \mu(R_{a,b}[y]) \rangle$. Let us define T_{ab} to be the transformation which maps an $\langle x_a, y_a \rangle$ pair represented with respect to the coordinate system of state a , to the same pair represented with respect to the coordinate system of state b , $\langle x_b, y_b \rangle$, (note that $T_{ab} = T_{ba}^{-1}$).

More explicitly, as before, let $\mu^\theta(a,b)$ be the mean change in heading from state a to state b (recall that $\mu^\theta(a,b) = -\mu^\theta(b,a)$). The transformation T_{ab} is defined as follows:

$$\begin{bmatrix} x_b \\ y_b \end{bmatrix} = T_{ab} \begin{bmatrix} x_a \\ y_a \end{bmatrix} = \begin{bmatrix} x_a \cos(\mu^\theta(a,b)) - y_a \sin(\mu^\theta(a,b)) \\ x_a \sin(\mu^\theta(a,b)) + y_a \cos(\mu^\theta(a,b)) \end{bmatrix}.$$

We can now redefine the consistency constraints for the x and y components of the odometric relation:

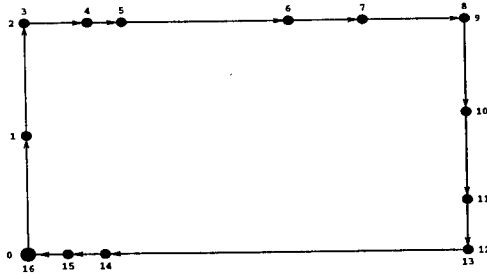


Figure 7: Model of a prescribed path through a true hallway environment.

- ◇ $\mu^{(x,y)}(a, a) = \langle 0, 0 \rangle$;
- ◇ $\mu^{(x,y)}(a, b) = -\mathcal{T}_{ba}(\mu^{(x,y)}(b, a))$ (*anti-symmetry*);
- ◇ $\mu^{(x,y)}(a, c) = \mu^{(x,y)}(a, b) + \mathcal{T}_{ba}(\mu^{(x,y)}(b, c))$ (*additivity*);

The reestimation formulae for all the parameters except for the x and y components of the relation matrix R , remain as before. However, the reestimation formulae for the x and y parameters are changed to reflect the relative coordinate systems used. $\mu_{i,j}^x$ and $\mu_{i,j}^y$ are reestimated as follows:

$$\begin{bmatrix} \bar{\mu}_{i,j}^x \\ \bar{\mu}_{i,j}^y \end{bmatrix} = \frac{\sum_{t=0}^{T-2} \xi_t(i, j) \begin{bmatrix} r_t[x] \\ r_t[y] \end{bmatrix} - \sum_{t=0}^{T-2} \xi_t(j, i) \mathcal{T}_{ji} \left(\begin{bmatrix} r_t[x] \\ r_t[y] \end{bmatrix} \right)}{\sum_{t=0}^{T-2} (\xi_t(i, j) + \xi_t(j, i))}.$$

These reestimation rules are guaranteed to satisfy the first two geometrical constraints, but not the additivity constraint. Their correctness can be proved along the lines of the correctness proofs for all other formulae [16].

5 EXPERIMENTS AND RESULTS

The goal of this work is to use odometry to improve the learning of topological models, while using fewer iterations and less data. We tested our algorithm in a simple robot-navigation world. In earlier stages of this work, a strong assumption underlay our experiments: the corridors in the environment are all perpendicular to each other, and the agent was using this perpendicularity to reset its position while accumulating the odometric readings. Here we have updated the algorithm and dropped the assumption. The experiments demonstrate that the use of odometry, even with accumulated rotational error and without using the perpendicularity assumption, is still very beneficial.

5.1 EXPERIMENTAL SETTING

Our experiments use both real robot data and simulated data. We ran our robot Ramona, a modified RWI B21, along a *prescribed*⁴ directed path in our department corridors. Low-level routines let Ramona move forward through

⁴Hence, no decisions are executed by the robot, and the model is an HMM and not a complete POMDP.

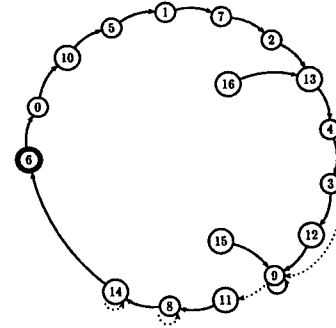


Figure 8: Learned topological model.

hallways from intersection to intersection and to turn ninety degrees to the left or right. Ultrasonic data interpretation let her perceive, in three directions – front, left and right – whether there is an open space, a door, a wall, or something unknown. Doors and intersections constitute *states*. When they are detected by Ramona, it stops and records its observations, as well as its odometric change between the previous and the current state. All recorded measures as well as the actions are, of course, subject to error.

The path Ramona followed consists of 4 connected corridors, which include 17 states, as shown in Figure 7. Black dots represent the physical locations of states. Multiple states (depicted as numbers in the plot) associated with a single location correspond to different orientations of the robot at that location. The larger black circle, at the bottom left corner, represents the starting position. The observations associated with each state are omitted for clarity. A projection of the odometric readings that Ramona recorded along the x and y dimensions, is shown in figure 5.

To statistically evaluate our algorithm, we use a simulated office environment in which the robot follows a prescribed path. It is represented as an HMM consisting of 44 states, and the associated transition, observation, and odometric distributions. Figure 9 depicts this HMM. Arrows represent transitions that have probability 0.2 or higher. Solid arrows represent the most likely transitions between the states. We generated 5 data sequences from the model, each of length 800, using Monte Carlo sampling. One of these sequences is depicted in Figure 10. Again, observations are omitted, and this is a projection of the odometry readings onto a global 2-dimensional coordinate system. For each sequence we ran our algorithm 10 times. We also ran the standard Baum-Welch algorithm, not using odometric information, 10 times on each sequence. For both algorithms we started each run from a randomly picked initial model.

5.2 RESULTS

We used our algorithm to learn a topological model of the environment from the data gathered by Ramona. Figure 8 shows the topology of one typical learned HMM. The bold circle represents the initial state. The arrows semantics is

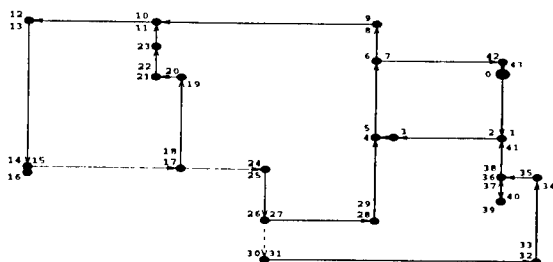


Figure 9: Model of a prescribed path through the simulated hallway environment.

as stated before. It is clear that the learned topology corresponds well to the topology of the true environment. The observation distributions learned are omitted from the figure, but they too correspond well to the walls, doors and openings encountered along the path, while incorporating the identification error resulting from noisy sensors.

Traditionally, in simulation experiments, learned models are quantitatively compared to the actual model that generated the data. Each of the models induces a probability distribution on strings of observations; the asymmetric Kullback-Leibler divergence [11] between the two distributions is a measure of how far the learned model is from the true model. We report our simulation results in terms of a sampled version of the KL divergence, as described by Juang and Rabiner [9]. It is based on generating sequences of sufficient length according to the distribution induced by the true model, and comparing their likelihoods according to the learned model with the true model likelihoods. We ignore the odometry information when applying the KL measure, thus allowing comparison between purely topological models that are learned with and without odometry.

Table 1 lists the KL divergence between the true and learned model, as well as the number of runs until convergence was reached, for each of the 5 simulation sequences under the two learning settings, averaged over 10 runs per sequence.

The table demonstrates that the KL divergence with respect to the true model for models learned using odometry, is about 4-5 times smaller than for models learned without odometric data. To check the significance of our results

Table 1: Average results of 2 learning settings with 5 training sequences.

Seq. #		1	2	3	4	5
With	KL	1.115	1.100	1.095	1.139	1.129
	Iter #	69.7	81.8	84.3	52.4	112.9
No	KL	5.575	4.499	4.997	4.491	5.791
	Iter #	120.4	107.5	116.2	113.3	120.6

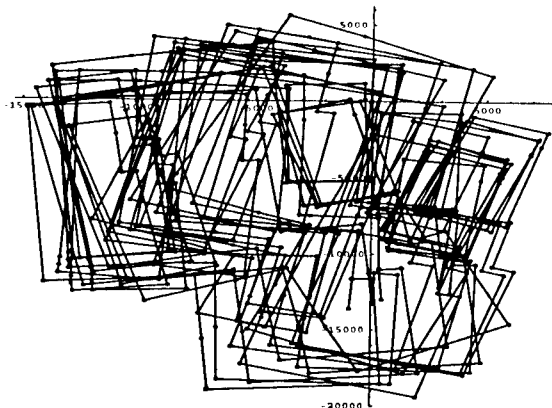


Figure 10: A data sequence generated by our simulator.

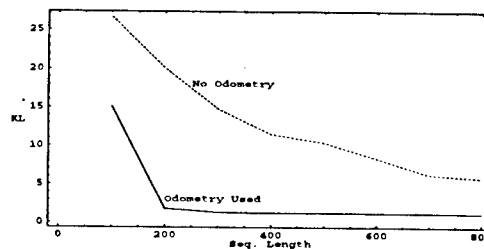


Figure 11: Average KL-divergence as a function of length.

we used the simple two-sample t-test. The models learned using odometric information have highly statistically significantly ($p \gg 0.9995$) lower average KL divergence than the others.

In addition, the number of iterations required for convergence when learning using odometric information is smaller than required when ignoring such information. Again, the t-test verifies the significance ($p > 0.995$) of this result.

To examine the influence of the amount of data on the quality of the learned models, we took one of the 5 sequences (Seq. #1) and used its prefixes of length 100 to 800 (the complete sequence), in increments of 100, as individual sequences. We ran the two algorithmic settings over each of the 8 prefix sequences, 5 times repeatedly. We then used the KL-divergence as described above to evaluate each of the resulting models with respect to the true model. For each prefix length we averaged the KL-divergence over the 5 runs. Table 2 summarizes the results of this experiment. It lists the mean KL-divergence over the 5 runs for each of the prefixes, as well as the standard deviation around this mean. The plot in Figure 11 depicts the KL-divergence as a function of the sequence length for each of the settings. Both the table and the plot demonstrate that, in terms of the KL-divergence, our algorithm, which uses odometric information, is robust in the face of data reduction. In contrast, learning without the use of odometry is much more sensi-

Table 2: Average results with 8 incrementally longer sequences.

Seq. Length		800	700	600	500	400	300	200	100
With	Mean KL	1.136	1.201	1.191	1.241	1.216	1.272	1.771	15.076
	Std. Dev.	0.091	0.083	0.131	0.082	0.036	0.085	0.510	12.884
No	Mean KL	5.790	6.249	8.354	10.390	11.490	14.772	20.044	26.619
	Std. Dev.	0.554	0.937	0.179	0.460	0.422	1.280	0.904	0.460

tive to reduction in the amount of data. Again, we applied the two-sample t-test, which verified the statistical significance of these results.

6 CONCLUSIONS

Directional information which comes up in various applications of computer science in general and machine learning in particular, requires special treatment. Currently most statistical models and applications are based on distributions that are either discrete or continuous along the real line, rather than circular. It is important to be aware of the need for circular distributions as well as of their existence. Moreover, it would be useful to have widely used applications such as Autoclass [2] support such distributions.

A problematic aspect of directional data which manifests itself when learning maps and models for robot navigation is that of cumulative rotational errors. In the context of our work we have demonstrated that the use of relative coordinate systems rather than global ones supports learning relationship between states. The main point shown by this paper is that through correct treatment of directional data, odometric information which is weak and very noisy still provides a significant leverage when learning a purely topological map.

Acknowledgments

We thank Sebastian Thrun for his insightful comments, and Dimitris Michailidis for his editorial help. This work was supported by DARPA/Rome Labs Planning Initiative grant F30602-95-1-0020, by NSF grants IRI-9453383 and IRI-9312395, and by the Brown University Graduate Research Fellowship.

References

- [1] A. R. Cassandra, L. P. Kaelbling, J. A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 1996.
- [2] P. Cheeseman, et al. Autoclass: A Bayesian classification system. In J. W. Shavlik, T. G. Dietterich, eds., *Readings in Machine Learning*. Morgan-Kaufmann, 1990.
- [3] A. P. Dempster, N. M. Laird, D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1), 1-38, 1977.
- [4] F. C. Dyer. Bees acquire route-based memories but not cognitive maps in a familiar landscape. *Animal Behaviour*, 41, 239-246, 1991.
- [5] Z. Ghahramani, M. I. Jordan. Factorial hidden Markov models. In *15th Int. Conf. on Machine Learning*. 1997.
- [6] E. G. Gumbel, J. A. Greenwood, D. Durand. The circular normal distribution: Theory and tables. *American Statistical Society Journal*, 48, 131-152, March 1953.
- [7] D. Heckerman, D. Geiger. Learning Bayesian networks: A unification for discrete and Gaussian domains. In *Proc. of the 11th Int. Conf. on Uncertainty in AI*. 1995.
- [8] B. H. Juang. Maximum likelihood estimation for mixture multivariate stochastic observations of Markov chains. *AT&T Technical Journal*, 64(6), July-August 1985.
- [9] B. H. Juang, L. R. Rabiner. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal*, 64(2), 391-408, February 1985.
- [10] S. Kotz, N. L. Johnson, eds. *Encyclopedia of Statistical Sciences*, vol. 2, pp. 381-386. John Wiley and Sons, 1982.
- [11] S. Kullback, R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1), 79-86, 1951.
- [12] F. Lu, E. E. Millios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4, 333-349, 1997.
- [13] K. V. Mardia. *Statistics of Directional Data*. Academic Press, 1972.
- [14] I. Nourbakhsh, R. Powers, S. Birchfield. Dervish: An office-navigating robot. *AI Magazine*, 16(1), 53-60, 1995.
- [15] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2), 257-285, February 1989.
- [16] H. Shatkay, L. P. Kaelbling. Learning hidden Markov models with geometric information. Tech. Rep. CS-97-04, Dept. of Computer Science, Brown University, 1997.
- [17] H. Shatkay, L. P. Kaelbling. Learning topological maps with weak local odometric information. In *Proc. of the 15th Int. Joint Conf. on AI*. 1997.
- [18] R. G. Simmons, S. Koenig. Probabilistic navigation in partially observable environments. In *Proc. of the Int. Joint Conf. on AI*. 1995.
- [19] R. Smith, M. Self, P. Cheeseman. A stochastic map for uncertain spatial relationships. In S. S. Iyengar, A. Elfes, eds., *Autonomous Mobile Robots*. IEEE Press, 1991.
- [20] S. Thrun, W. Burgard, D. Fox. A probabilistic approach to concurrent map acquisition and localization for mobile robots. *Machine Learning*, 31, 29-53, 1998.

A Neural Network Model for Prognostic Prediction

W. Nick Street

Computer Science Department
Oklahoma State University
Stillwater, OK 74078
nstreet@cs.okstate.edu

Abstract

An important and difficult prediction task in many domains, particularly medical decision making, is that of prognosis. Prognosis presents a unique set of problems to a learning system when some of the outputs are unknown. This paper presents a new approach to prognostic prediction, using ideas from nonparametric statistics to fully utilize all of the available information in a neural architecture. The technique is applied to breast cancer prognosis, resulting in flexible, accurate models that may play a role in preventing unnecessary surgeries.

1 Introduction

This paper applies artificial neural network classification to the analysis of *survival* or *lifetime data* (Lee, 1992), in which the objective can be broadly defined as predicting the future time of a particular event. In this work we are concerned specifically with prognosis, that is, predicting the course of a disease. These methods are applied to breast cancer prognosis, predicting how long after surgery we can expect the disease to recur. This problem has significant clinical importance. Decisions regarding chemotherapy its intensity are based on the anticipated course of the cancer. For example, patients with favorable outlooks may forego chemotherapy entirely. Those with less favorable outlooks may undergo varying intensities of chemotherapy, or even bone marrow transplantation.

Prognostic prediction does not fit comfortably into either of the classic learning paradigms of function approximation or classification. While a patient can be classified "recur" if the disease is observed, there is

no real cutoff point at which the patient can be considered a non-recurrent case. The data are therefore *censored* in that we know a time to recur for only a subset of patients. For the others, we know only the time of their last check-up, or disease-free survival time (DFS). In particular, recurrence or survival data is *right censored*, i.e., the right endpoint (recurrence time) is sometimes unknown, since some patients will inevitably move away, change doctors, or die of unrelated causes. Therefore, in many cases, the training signal for the learning method is not well-defined. Prognosis is not viewed here as a time-series prediction problem, since the predictive features are gathered only once, at the time of diagnosis and/or surgery.

Problems involving censored data are common to several fields. In engineering, one might be interested in the survival characteristics of electronic components, while sociologists might consider what factors lead to long-lasting marriages. These problems have traditionally been approached using statistical techniques such as Cox proportional-hazards regression (Cox, 1972). In recent years, there has been an increased interest in the application of machine learning methods to prediction using censored data. Several groups have approached prognosis as a separation problem using different learning architectures, including backpropagation artificial neural networks (ANNs) (Burke, 1994; Burke et al., 1997), entropy maximization networks (Choong et al., 1996) and decision trees (Wolberg et al., 1992; Wolberg et al., 1994). This is done by choosing one or more endpoints and learning a yes/no classifier on concepts such as "patients who recurred in less than two years." Cases with follow-up time less than the cutoff are discarded from the training set. Ravdin and colleagues (De Laurentiis and Ravdin, 1994; Ravdin and Clark, 1992) use ANNs to generate *survival curves*, which plot the probability of disease-free survival against time. This work uses time

as an input variable and interprets the trained network's single output as an approximation of recurrence probability. The resulting formulation results in biases in the training data that must be corrected by repeating or removing some of the examples. Their computational results are verified only by demonstrating that their predicted survival rates closely approximate those of the test cases. The problem has also been approached in an unsupervised learning fashion, using clustering (Bradley et al., 1997) and self-organizing neural networks (Schenone et al., 1993). However, these techniques did not directly address the problem of prediction using censored data.

While this research also separates the cases into classes based on recurrence time, it differs from the above techniques in several respects. Censored cases are incorporated directly into the training set, not by using an artificial cutoff time, but rather by using the probability that they will recur before a certain time as the training signal. In this way we use all of the information available in the training set. Further, interpreting the outputs as probabilities lets us not only separate the cases into "good" and "bad" prognoses, but also to generate predicted survival curves for individual patients, making the system more useful in a clinical setting.

2 Neural Architecture

The ANNs used in this work were standard feedforward networks with one hidden layer, trained with backpropagation (Rumelhart et al., 1986). The hyperbolic tangent activation function was used for hidden and output nodes. The output layer consisted of ten units; the first represented the class of examples with recurrences at one year or less following surgery, the second those with recurrences between one and two years, etc., up to ten years¹. This approach implies the existence of an extra (in our case, eleventh) class. These are the patients with expected disease-free survival of time greater than the length of the study (10 years). The activations of the output units were trained with and interpreted as the probability that the patient would have disease-free survival up to that time. These probabilities were scaled to the range of the hyperbolic tangent function, i.e., $activation = 2 * probability - 1$.

In order to maintain the interpretation of the outputs as probabilities, the *relative entropy* error func-

tion (Baum and Wilczek, 1988; Solla et al., 1988) was used for all non-input units. For a given example i , this error function is defined as

$$E_i = \sum_k \left[\frac{1}{2} (1 + T_i^k) \log \frac{1 + T_i^k}{1 + O_i^k} + \frac{1}{2} (1 - T_i^k) \log \frac{1 - T_i^k}{1 - O_i^k} \right],$$

where T_i^k is the target value for output unit k and O_i^k is its output value. Outputs of +1 and -1 correspond to definitely true and definitely false, respectively, with intermediate values again being scaled into the appropriate range.

For recurrent cases, the network was trained with values of +1 for all outputs up to the observed recurrence time, and -1 thereafter. For instance, a recurrence at 32 months would have a training vector $T = \{1, 1, -1, -1, -1, -1, -1, -1, -1\}$. The value of the probability formulation is seen in the censored cases. They were similarly trained with values of +1 up to the observed disease-free survival time. The probabilities of DFS for later times were computed using a variation of the standard Kaplan-Meier maximum likelihood approximation to the true population survival rate (Kaplan and Meier, 1958). We define the *risk* of recurrence at time $t > 0$ as the conditional probability that a patient will recur at time t , given that they have not recurred up to time $t - 1$. As an example, consider a study containing a total of 20 patients. If two recurrences were observed in the first time interval, we would have $risk_1 = 0.1$. Further suppose that the study has two censored cases in the first time interval, and two more recurrences in the second interval. There are 16 patients at risk for recurrence during interval two, with two recurrences, so $risk_2 = 0.125$. The Kaplan-Meier estimator of the *disease-free survival curve*, S , tracks the cumulative probability of DFS for any time in the study, using the risks in the following fashion:

$$S_t = \begin{cases} 1, & t = 0 \\ S_{t-1}(1 - risk_t), & t > 0. \end{cases}$$

Continuing the above example, $S_0 = 1.0$, $S_1 = 0.9$, and $S_2 = 0.7875$. To compute appropriate training probabilities, we simply use the DFS time of the censored case as the starting time, rather than time 0:

$$S_t = \begin{cases} 1, & 0 \leq t \leq DFS(i) \\ S_{t-1}(1 - risk_t), & t > DFS(i). \end{cases}$$

For an individual output node k , this training signal represents the example's probability of membership in

¹The available prognostic studies are approximately ten years in duration.

the class being recognized by that node, i.e., the set of cases that recur before the end of year k . Collectively, the activation values of the output units represent an expected survival curve for the individual case.

If we view the network as learning a survival curve, the task becomes one of function approximation using incomplete data. The training signal is then a modified thermometer encoding (McCullagh and Nelder, 1989), a relatively common encoding for ordered categorical outputs, with the added complication of the survival probabilities for censored cases. Since the effects of some of the input features are thought to be nonlinear over time, it is also instructive to view the problem as a sequence of highly related but distinct classification problems, all learned using the same internal representation (i.e., hidden nodes). The representation generated in learning one group (say, those cases that are likely to recur before one year) contributes to the learning of other groups (say, those cases recurring between 5 and 6 years). This is a form of functional knowledge transfer, similar to the MTL network (Baxter, 1995; Caruana, 1995). The learning of multiple classes in parallel contributes to faster learning and more reliable predictive models.

The above architecture facilitates three different uses of the resulting predictive model:

1. The output units can be divided into groups *a posteriori* to separate good from poor prognoses. For a particular application, any prediction of recurrence at a time greater than five years might be considered favorable, and indicate less aggressive treatment. The actual outcomes of those patients in the good group should be significantly better than those in the poor group.
2. An individualized disease-free survival curve can easily be generated for a particular patient by plotting the probabilities predicted by the various output units. In order for this curve to be reliable, the activations should be monotonically decreasing, or very nearly so.
3. The expected time of recurrence can be obtained merely by noting the first output unit that predicts a probability of disease-free survival of less than 0.5. This provides a convenient method of rank-ordering the cases according to expected outcome.

A significant methodological issue is that of evaluating the learned model. As discussed earlier, this is neither

a function approximation nor a classification problem, since in many cases we do not know the correct answer. Still, there is a well-defined goal: the accurate prediction of individual prognosis. While our training method seeks to minimize the relative entropy error at each output unit, the reporting of this error on testing sets would be relatively uninformative. We therefore evaluate the models on two criteria: the accuracy of the predicted recurrence rates (see Section 3.4) and the ability of the models to separate cases with favorable and unfavorable prognoses (see Section 3.3).

3 Experimental Results

Computational experiments were performed on two very different breast cancer data sets. The first is known as Wisconsin Prognostic Breast Cancer (WPBC) and is characterized by a small number of cases, relatively high dimensionality, very precise values and almost no missing data. The second data set is from the Surveillance, Epidemiology, and End Results (SEER) program of the National Cancer Institute. It contains a large number of cases, with relatively few, coarsely-measured features, and a high percentage of missing values. Details on these data sets are given below.

In both cases, the prognosis data used in this study consists of those malignant patients for which follow-up data was available, after eliminating those cases with distant metastasis (cancer has already spread; prognosis is poor) and carcinoma *in situ* (cancer has not yet invaded breast tissue; prognosis is good). We therefore maximize the clinical relevance of the study by focusing on those cases that present the most difficult prognosis.

Experiments reported in this section are test set results using either tenfold cross-validation (WPBC data) or a single randomized holdout test (SEER data). The ANNs used had three hidden units, and training was terminated after 1000 on-line epochs.

3.1 Wisconsin Prognostic Breast Cancer Data

In previous work (Mangasarian et al., 1995; Wolberg et al., 1994) the author contributed to the development of an image-processing software package for breast cancer diagnosis, known as Xcyt, which analyzes digital images of cells taken from breast lumps. This program computes 10 different features of each cellular nuclei in the image: radius, perimeter, area, compactness,

smoothness, size and number of concavities, symmetry, fractal dimension, and texture. The mean, standard area, and extreme values of each feature are computed for each image. The current application uses the 30 nuclear features computed by Xcvt together with two traditional prognostic predictors: tumor size and number of involved lymph nodes. This data set contains 227 cases, 61 of which have recurred. An earlier version of this data set is available at the UCI machine learning repository (Merz and Murphy, 1996).

3.2 SEER Data

The SEER (Carter et al., 1989) data set consists of data on cancer survival (rather than recurrence) for over 38,000 women newly diagnosed with breast cancer between 1977 and 1982. Each case contains the following information: histological grade (four discrete values), tumor size, tumor extent (5 discrete values), number of positive lymph nodes, and number of nodes examined. Many of these feature values are missing. For instance, only about 20% of the cases contain a value for histological grade; over 1200 of the cases contain no feature information at all. Each of the SEER features was encoded as a sequence of binary variables, with an additional binary variable representing a missing value.

3.3 Good vs. poor prognoses

To be used as a clinical tool, the predictive model should reliably separate cases with a good prognosis from those with a poor prognosis. Since treatment options are limited, this sort of stratification could be most helpful to the physician and the patient in determining a post-operative treatment plan. Figure 1 stratifies the WPBC test cases into those predicted to recur in the first five years and those predicted to recur at some time greater than five years (including the implicit 11th class). The difference in these two groups is statistically significant ($p < 0.001$, generalized Wilcoxon test). Of course, the output units could be grouped differently to define the relevant prognostic categories for a particular problem. Further, the implicit final group could also be subdivided based on the activation level of the last node.

Similarly, Figure 2 shows survival probabilities for those cases with good and poor prognosis, in this case, predicted survival less than or equal to ten years and predicted survival greater than ten years. Again the difference in the two groups is statistically significant ($p < 0.001$). The difference in dividing points between

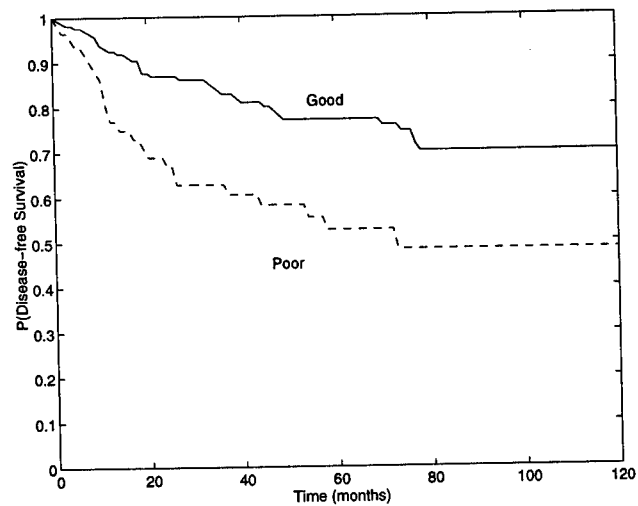


Figure 1: WPBC Data: Disease-free survival probabilities for those cases predicted to recur in the first five years (Poor, 58 cases) compared to those predicted to recur at some time greater than five years (Good, 169 cases).

the two tests is due to the difference between the measured endpoints (recurrence in the WPBC data, death in the SEER data). The ratios of good to bad prognoses were held nearly constant.

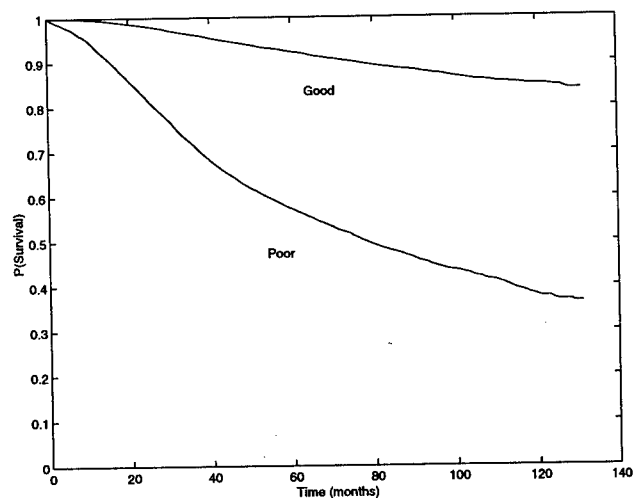


Figure 2: SEER Data: Survival probabilities for those cases predicted to die from breast cancer in the first ten years (Poor, 8,353 cases) compared to those predicted to die at some time greater than ten years (Good, 26,192 cases).

In traditional breast cancer staging, postoperative treatment decisions are based largely or even entirely on whether or not the cancer has spread to the patient's axillary lymph nodes. However, removing the nodes for examination leaves the arm subject to infection and possible lymphedema (Aitken et al., 1989), and does not affect overall survival (Abe et al., 1995). In both of our test cases, the separation with this method represents an improvement over that achieved by the lymph node status feature. Further, statistically significant separation was achieved in both data sets without using the lymph feature (WPBC, $p = 0.02$; SEER, $p < 0.001$). This is further confirmation of a previous finding (using other analytic techniques) that breast cancer prognosis can be achieved without lymph node dissection (Wolberg et al., 1997; Wolberg et al., 1998).

3.4 Predicted vs. actual group survival

Another criterion for the validity of the learned model is whether the predicted recurrence rate follows that of the actual data. Figure 3 shows the Kaplan-Meier estimate of disease-free survival curve for the entire WPBC training set, compared with the predicted DFS rates accumulated from the test folds. Again, a test case is predicted to recur at time t if the activation of output node t is the first one indicating a DFS probability of less than 0.5. The two curves are very similar and show no significant statistical difference ($p = 0.2818$, generalized Wilcoxon test (Gehan, 1965)).

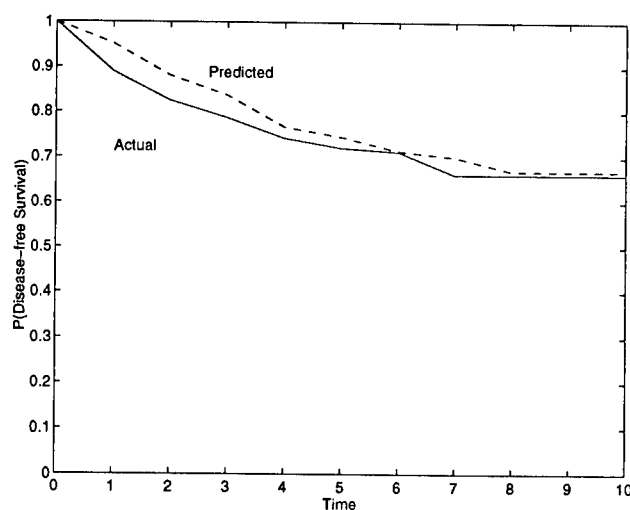


Figure 3: WPBC Data: Kaplan-Meier estimate of true disease-free survival curve compared to predicted DFS curve.

The predicted group survival for the SEER data did not closely match the actual survival curve. This is consistent with previous research (Street et al., 1996) using a variation of the RSA prognostic technique (Street et al., 1995) which also showed that the SEER data was unable to replicate group survival characteristics. This is attributable to the coarse encoding of the SEER variables and the large percentage of missing values.

3.5 Individual prognostic prediction

As mentioned, the activations of the output units can be combined to form a predicted DFS curve for an individual patient. Figure 4 shows an example of this usage, in a format appropriate for a clinical setting. Here the probabilities of disease-free survival for a case from the WPBC study are compared to the cumulative values of all patients in the study. The output activations were monotonically non-increasing, as was the case in 74% of the WPBC test examples and 87% of the SEER examples. The others had occasional small increases, with the maximum increase in any example corresponding to a probability change of 0.024 (WPBC) and 1.0 (SEER). The expected time of recurrence can be computed by noting where the DFS curve crosses a probability of 50%, in this case, between three and four years. In fact, this patient did experience disease recurrence in the 44th month following surgery.

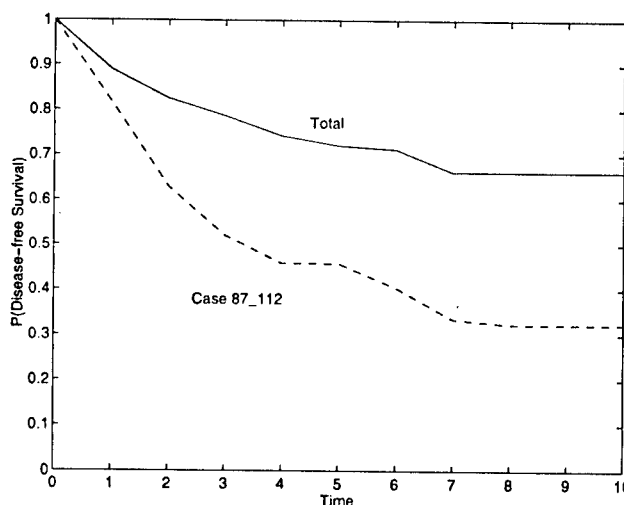


Figure 4: Predicted DFS curve of a single case (87.112) from the WPBC study compared to the overall group DFS curve of the training set.

4 Conclusions

This paper develops a novel encoding of censored data in an artificial neural network architecture to provide a framework for prognostic prediction. In applying the method to breast cancer prognosis, the resulting models are shown to be at least as accurate as current methods, while providing significantly more precision and flexibility. Among the future directions for this research is a sensitivity analysis to investigate the importance of the prognostic features at different follow-up times. To evaluate the role of knowledge transfer, predictive accuracy will be compared to classification models that predict recurrence at a chosen cut point. Most importantly from a clinical perspective, our work in the breast cancer domain continues to focus on generating accurate prognostic models without knowledge of lymph node status, in order to spare new patients an extra and potentially debilitating surgery.

Acknowledgments

The author wishes to thank Dr. William H. Wolberg of the University of Wisconsin Department of Surgery for contributing both data and direction to this research. The SEER data was made available by Dr. Don Henson of the National Cancer Institute. This work was partially supported by NSF grant IRI-9701992.

References

- Abe, O., Abe, R., Asaishi, K., Enomoto, K., Hattori, T., and Iino, Y. (1995). Effects of radiotherapy and surgery in early breast cancer: An overview of the randomized trials. *New England Journal of Medicine*, 333:1444-1455.
- Aitken, R. J., Gaze, M. N., Rodger, A., Chetty, U., and Forrest, A. P. M. (1989). Arm morbidity within a trial of mastectomy and either nodal sample with selective radiotherapy or axillary clearance. *British Journal of Surgery*, 76:568-571.
- Baum, E. B. and Wilczek, F. (1988). Supervised learning of probability distributions by neural networks. In Anderson, D. Z., editor, *Neural Information Processing Systems*, pages 52-61, New York. American Institute of Physics.
- Baxter, J. (1995). Learning internal representations. In *Proceedings of the Eighth International Conference on Computational Learning Theory*.
- Bradley, P. S., Mangasarian, O. L., and Street, W. N. (1997). Clustering via concave minimization. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, pages 368-374, Cambridge, MA. MIT Press.
- Burke, H. B. (1994). Artificial neural networks for cancer research: Outcome prediction. *Seminars in Surgical Oncology*, 10:73-79.
- Burke, H. B., Goodman, P. H., Rosen, D. B., Henson, D. E., Weinstein, J. N., Harrell, F. E., Marks, J. R., Winchester, D. P., and Bostwick, D. G. (1997). Artificial neural networks improve the accuracy of cancer survival prediction. *Cancer*, 79:857-862.
- Carter, C. L., Allen, C., and Henson, D. E. (1989). Relation of tumor size, lymph node status, and survival in 24,740 breast cancer cases. *Cancer*, 63:181-187.
- Caruana, R. (1995). Learning many related tasks at the same time with backpropagation. In *Advances in Neural Information Processing Systems*, volume 7, pages 657-664.
- Choong, P. L., deSilva, C. J. S., Dawkins, H. J. S., and Sterrett, G. F. (1996). Entropy maximization networks: An application to breast cancer prognosis. *IEEE Transactions on Neural Networks*, 7(3):568-577.
- Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society, B* 34:187-202.
- De Laurentiis, M. and Ravdin, P. M. (1994). A technique for using neural network analysis to perform survival analysis of censored data. *Cancer Letters*, 77:127-138.
- Gehan, E. A. (1965). A generalized Wilcoxon test for comparing arbitrarily single-censored samples. *Biometrika*, 52:203-223.
- Kaplan, E. L. and Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457-481.
- Lee, E. T. (1992). *Statistical Methods for Survival Data Analysis*. John Wiley and Sons, New York.
- Mangasarian, O. L., Street, W. N., and Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570-577.

- McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, chapter 6, page 137. Chapman and Hall, London, 2 edition.
- Merz, C. J. and Murphy, P. M. (1996). UCI repository of machine learning databases [<http://www.ics.uci.edu/~mlearn/MLRepository.html>]. University of California, Irvine, Department of Information and Computer Sciences.
- Ravdin, P. M. and Clark, G. M. (1992). A practical application of neural network analysis for predicting outcome of individual breast cancer patients. *Breast Cancer Research and Treatment*, 22:285-293.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1, chapter 8. MIT Press, Cambridge, MA.
- Schenone, A., Andreucci, L., Sanguinetti, V., and Morasso, P. (1993). Neural networks for prognosis in breast cancer. *Physica Medica*, IX(Supplement 1):175-178.
- Solla, S. A., Levin, E., and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, 2:625-639.
- Street, W. N., Mangasarian, O. L., and Wolberg, W. H. (1995). An inductive learning approach to prognostic prediction. In Prieditis, A. and Russell, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 522-530, San Francisco. Morgan Kaufmann.
- Street, W. N., Mangasarian, O. L., and Wolberg, W. H. (1996). Individual and collective prognostic prediction. Technical Report 96-01, Computer Sciences Department, University of Wisconsin, Madison, WI. Available from <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/>.
- Wolberg, W. H., Bennett, K. P., and Mangasarian, O. L. (1992). Breast cancer diagnosis and prognostic determination from cell analysis. Manuscript, Departments of Surgery and Human Oncology and Computer Sciences, University of Wisconsin.
- Wolberg, W. H., Street, W. N., and Mangasarian, O. L. (1994). Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Letters*, 77:163-171.
- Wolberg, W. H., Street, W. N., and Mangasarian, O. L. (1997). Computer-derived nuclear features compared with axillary lymph node status for breast cancer prognosis. *Cancer Cytopathology*, 81:172-179.
- Wolberg, W. H., Street, W. N., and Mangasarian, O. L. (1998). A comparison of computer-based nuclear analysis versus lymph node status for staging breast cancer. *Lancet*. submitted.

Learning the Grammar of Dance

Joshua M. Stuart

Department of Computer Science
University of Colorado
Boulder CO 80309-0430 USA

Elizabeth Bradley*

Department of Computer Science
University of Colorado
Boulder CO 80309-0430 USA

Abstract

A common task required of a dancer or athlete is to move from one prescribed body posture to another in a manner that is consistent with a specific style. One can automate this task, for the purpose of computer animations, using simple machine-learning and search techniques. In particular, we find kinesiological and stylistically consistent interpolation sequences between pairs of body postures using graph-theoretic methods to learn the "grammar" of joint movements in a given corpus and then applying memory-bounded A* search to the resulting transition graphs — using an influence diagram that captures the topology of the human body in order to reduce the search space.

1 INTRODUCTION

A common task required of a dancer or athlete is to move from one prescribed body posture to another in a manner that is consistent with a specific style. If these postures are "far apart," as measured by some metric that takes into account both the kinesiology of the body and the style of the movement genre, this can be nontrivial. For the purposes of computer-generated animation, there are a variety of ways to generate movement sequences that accomplish this kind of task. One can, for instance, use mathematical interpolation techniques like splines to move individual body parts from one position to another, but these kinds of methods do not address the problem of kinesiological illegality (e.g., that the knee only bends 180 degrees, or that arms cannot pass through ribcages). Many animation packages, such as Life Forms (<http://fas.sfu.ca/lifeforms.html>), use an augmented spline approach that relies on a

table of kinematic constraints to avoid illegal movements, but this type of approach is somewhat *ad hoc*. A more-general way is to use the physics of the body: derive the associated differential equations — a torque balance for each joint, say — and solve the equivalent boundary-value problem. Approaches like this (Hodgins *et al.* 1995) are extremely interesting and highly promising, but also very difficult; deducing the control equations that humans use to recover their balance after a jump, for example, is a Ph.D. thesis-level problem (Wooten 1998). *Stylistically* faithful interpolations would be even harder to implement; neither splines nor $F = ma$ can easily capture or enforce, for instance, the requirement that classical ballet emphasizes position over motion¹, and developing a mathematics- or physics-based approach that does so would be all but impossible. In this paper, we propose an alternative solution to this problem: a class of corpus-based interpolation schemes that generate a kinesiological and stylistically consistent movement sequence between two specified body positions by learning and then enforcing the dynamics of a particular movement genre.

The primary motivation for the development of these methods was our work on a mathematical technique (Bradley & Stuart 1997; 1998) that automatically creates variations on predefined motion sequences — an idea that was inspired by a similar scheme (Dabby 1996; 1997) that uses a related procedure to generate musical variations. We use the mathematics of nonlinear dynamics to shuffle a predefined movement sequence by "wrapping" a progression of special symbols representing the body positions in a dance piece, martial arts form, or other motion sequence around a chaotic attractor. This establishes a symbolic dynamics that links the movement pro-

¹In ballet, body parts tend to describe piecewise-linear paths through space, emphasizing the positions at the junctions of those linear segments; in modern dance, on the other hand, the motion *between* the endpoints is the important feature.

*Author to whom correspondence should be sent

gression and the attractor geometry, as shown in figure 1. By definition, trajectories from different starting points² travel along the same attractor but *in a different order*. This property lets us use the mapping depicted in figure 1(d) to create a variation: we simply follow a *new* trajectory around the attractor and invert the symbolic mapping, “playing” the body position for each cell the trajectory enters. Variations generated in this manner, whether musical or choreographic, are both aesthetically pleasing and strikingly reminiscent of the original sequences. The stretching and folding of the chaotic dynamics guarantee that the ordering of the pitches or movements in the variation is different from the original sequence; at the same time, the fixed geometry of the attractor ensures that a chaotic variation of Bach’s Prelude in C Major or of a short Balanchine ballet sequence are related to the original piece in a sense reminiscent of the classic “variation on a theme.” Broadly speaking, the chaotic variations resemble the originals with some shuffling of coherent subsequences. This is the primary source of the stylistic originality of the chaotic variation scheme — in fact, this type of subsequence shuffling is a well-established creative mechanism in modern choreography. One problem with any choreographic technique, automated or not, that involves subsequence reordering, however, is that the transitions at the subsequence boundaries can be quite jarring. Figure 2, for example, shows a short section of a chaotically generated variation on a short ballet adagio. Note the abrupt transition between the fifth and sixth moves of the variation.

The interpolation algorithms that are the topic of this paper can smooth these kinds of transitions in a manner that is both kinesiological and stylistically consistent. These graph-theoretic methods “learn” the grammar of joint movements in a given corpus and then apply memory-bounded A* search — using an influence diagram that models the relationships of the joints in the human body in order to reduce the otherwise-intractable search space — to find an appropriate interpolation sequence between two given body positions. The search is complicated by the fact that joint positions cannot be interpolated in isolation: the movement patterns of the ankle, for instance, are strongly influenced by whether or not the foot is on the ground — information that is implicit in the positions of the pelvis, knees, etc. This requires that the expansion of nodes in the search be context dependent in a somewhat unusual way. The resulting interpolation procedures, which were developed and evaluated in close collaboration with several expert dancers, are quite effective at capturing and enforcing the dynamics of a given group of movement sequences.

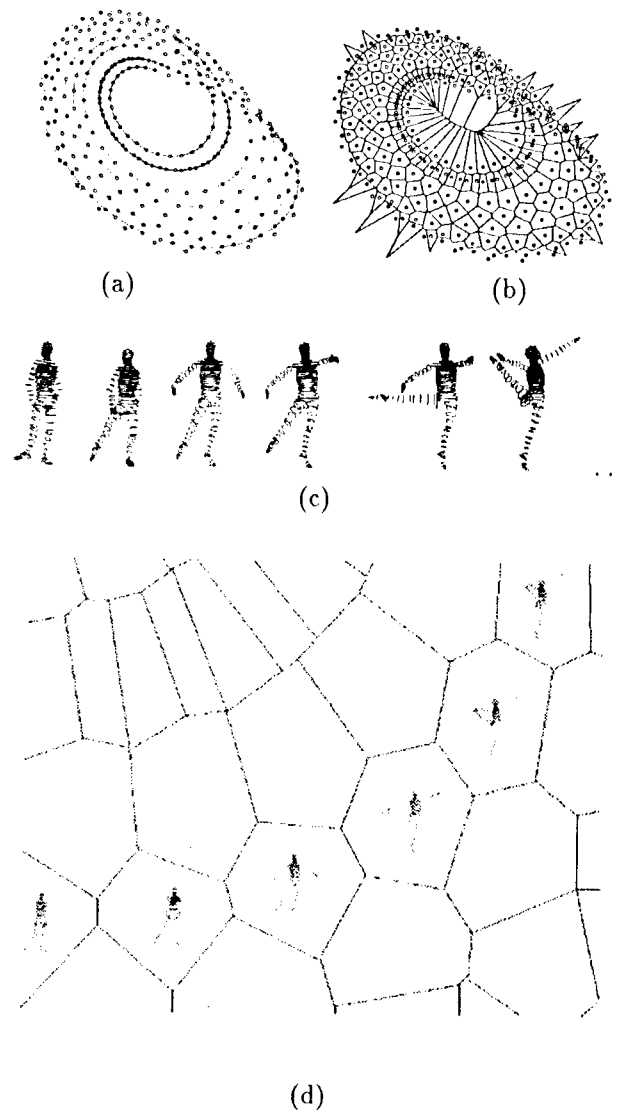


Figure 1: A chaotic mapping that links a short ballet adagio and the chaotic Rössler attractor. A Voronoi diagram is used to divide the region covered by the trajectory shown in part (a) into cells, yielding the tiling shown in part (b). The order in which the original trajectory traverses those cells defines the temporal order of the *cell itinerary* that corresponds to that trajectory. Successive body positions from the predefined movement sequence (c) are mapped to successive cells in that itinerary, linking the structure of the movement sequence and the attractor geometry. A small section of the overall mapping is shown in part (d).

²within the basin of attraction, of course



Figure 2: Part of a variation on a short ballet sequence, generated using the chaotic shuffling procedure diagrammed in the previous figure. Note the abrupt transition between the fifth and sixth frames. The interpolation schemes described in this paper can be used to smooth such transitions in a kinesiologically and stylistically consistent fashion.

2 CORPUS-BASED INTERPOLATION ALGORITHMS FOR MOVEMENT SEQUENCES

The interpolation schemes described in this section use corpora of human movement — a corpus composed of ten Balanchine ballets, for instance, if one is working with dances of that particular genre³ — to select a movement sequence that would naturally occur between a given pair of body postures. The basic algorithms involved are fairly straightforward, but the application requires some unusual tactics and variations. We first examine the corpus, capturing typical progressions of joint positions in a set of transition graphs. Then, given a pair of body postures, we use a variant of the A* algorithm to search these graphs for interpolation subsequences. A typical interpolation sequence might, for instance, first move the shoulder from its position in the fifth frame of figure 2 to its position in the sixth frame according to the rules for *shoulder* movement that are implicit in the corpus, then repeat for the elbow, and so on.

Our original approach (Bradley & Stuart 1997) was much more coarse-grained; the atomic representational unit was a full body position and the patterns in the corpus were represented in a single graph that had one vertex for each observed posture. This approach was both impractical and unsatisfying. Firstly, it did not scale well with corpus size because the number of unique body positions is so large. Secondly, it could only populate interpolation sequences with verbatim copies of full-body positions that appeared in the corpus. The methods described in this paper, on the other hand, construct the body positions in the interpolation

sequence in a joint-wise manner and on the fly. This scheme not only avoids the storage problems of the previous approach, but also allows innovation: it can generate sequences that contain body positions that do not appear in the corpus.

2.1 BODY POSTURE REPRESENTATION

We represent a human body posture by specifying the position of each of the 23 main joints with a *quaternion*, a standard representation in rigid-body mechanics that dates back to Hamilton (Goldstein 1980). A quaternion $q = (r, \vec{u})$ consists of an axis of rotation \vec{u} and a scalar r that specifies the angle of rotation of the joint about \vec{u} . Thus, a body-position symbol is quite complicated: 23 descriptors (*pelvis*, *right-wrist*, etc.), 92 numbers (four for each joint), and a variety of information about the position and orientation of the center of mass.

Joint orientations are, in reality, continuous variables, but computational complexity requires that they be discretized in our algorithms. Specifically, each joint λ can take on a finite number M^λ of allowed orientations⁴. Formally, we define Q^λ as the set of *allowed* orientations for joint λ and then replace the *actual* orientation of the joint with the closest quaternion in Q^λ . We can express a body position \vec{b} as a discretized vector \vec{s} by setting each of its components s_λ equal to the quaternion in Q^λ that is closest to b_λ : $s_\lambda = r$ such that $\|b_\lambda - r\| \leq \|b_\lambda - q\|$ for all $q, r \in Q^\lambda$ where $\|x - y\|$ is the Euclidean distance⁵ between the quaternions x and y . We can find r in $\log(M^\lambda)$ time using K-D trees (Friedman, Bentley, & Finkel 1977) to represent the Q^λ sets. The procedure described in this paragraph is analogous to “snapping” objects to a grid in computer drawing applications.

Deriving a successful discretization of joint states

⁴In practice, $M^\lambda < 400$.

⁵One of the main advantages of quaternions is that they can be treated as 4-vectors in the standard norm and transformation operations.

³The composition of the corpus will, of course, affect the nature of the interpolation; smoothing abrupt transitions in ballet pieces using an interpolation scheme that is mathematically rooted in a karate corpus will negate the very aesthetic resemblance that this approach strives to preserve. On the other hand, this might be an interesting source of innovation, whereby one could mathematically mix two or more styles.

was unexpectedly difficult. Simply discretizing the quaternion variable values — that is, classifying all positions between, say, (**right-wrist**, 1, 1, 0, 1) and (**right-wrist**, 1, 1, 0.2, 1) as an equivalence class and representing them in the algorithms as a single posture — produced visibly awkward animations. The human visual perception system appears to be very sensitive to small variations in quaternion coefficients: any change in a single coefficient seems to violate the “motif” of the motion. The same problem arose when we attempted a physically more-realistic discretization by transforming quaternion data to Euler angles and then discretizing θ , ϕ , and ψ instead. The solution on which we eventually settled uses a discretization library that was created by hand by an expert dancer.

2.2 REPRESENTATION OF A MOVEMENT CORPUS

2.2.1 Joint Transition Graphs

A transition graph is a weighted-directed graph that captures the transition probabilities in a symbol sequence. In general, each vertex v in such a graph represents a symbol and each weighted edge (v, u) reflects the probability that the symbol associated with vertex u follows the symbol associated with vertex v . For the purposes of analyzing a human movement corpus, we build one transition graph for each joint, using the corpus to identify orientations that the joint assumes and to estimate the corresponding transition probabilities. Vertices in this kind of graph represent particular discretized joint orientations, and edges correspond to the movement of the joint from one orientation to another.

The transition graph construction procedure is fairly straightforward. We first transform every body position in the corpus to a discretized position, as described in the previous section, so that a consecutive pair of body positions (\vec{a}, \vec{b}) , each consisting of 23 continuous-valued quaternions, becomes the discretized pair (\vec{s}, \vec{t}) where \vec{s} , \vec{t} each consist of 23 discretized quaternions. We then build a transition graph G^λ for each joint λ ; G^λ contains M^λ vertices, each of which corresponds to exactly one quaternion in Q^λ . For convenience, we will refer to vertices in G^λ by the corresponding quaternions in Q^λ . We record the fact that joint λ is allowed to move from a_λ to b_λ by introducing an edge in G^λ from vertex s_λ to vertex t_λ . We assign a weight to this edge that models the “unlikelihood” with which such a transition occurs in the corpus. This measure of unlikelihood is related to $P(q \rightarrow r)$, the probability that joint λ moves from the quaternion $q \in Q^\lambda$ to the quaternion $r \in Q^\lambda$, per the following expression for the weight of edge $(q, r) \in G^\lambda$:

$$w_{q,r}^\lambda = -\log(P(q \rightarrow r)) = -\log(P(r|q))$$

$$\approx \log(C(q)) - \log(C(q, r))$$

where $C(q)$ is the number of times joint λ assumed an orientation approximated by q and $C(q, r)$ is the number of times that the ordered pair (q, r) occurred. Larger weights correspond to transitions that are less likely to occur⁶.

Figure 3 shows a transition graph for the hips that was constructed in this fashion from a corpus of 38 short ballet sequences totaling 1720 positions. In the interests of clarity, edge weights and isolated vertices have been omitted from this figure. The intricate patterns in these dance progressions are reflected by the complex topology of the graph.

2.2.2 Coordinating Joint Movements

A joint transition graph represents the behavior of a joint *in isolation*. This information, alone, cannot capture the physical constraints that govern the coordination of the joints in the body. For example, if the shoulder is in its resting position with the palm facing the thigh, the elbow can bend nearly 180 degrees, but if the shoulder is turned 90 degrees on its long axis (until the palm faces backwards), the elbow can only bend about five degrees before the hand collides with the leg. In order to construct sensible interpolation sequences, we need a simple and efficient model of this type of joint coordination.

The most complete and general approach to this problem would be to model the interactions between each joint and every other joint in the body, but doing so engenders a combinatorial explosion in the search space. There are sensible ways to reduce the complexity of the problem, however; to a first approximation, a joint is not influenced by every other joint in the body. The position of the wrist, for instance, strongly affects the position of the fingers but has little effect on the toes. We put this simplifying assumption into effect by using an *influence diagram* (Oliver & Smith 1990) that reflects the structure and physics of the human body to explicitly represent the relationships of the joints to one another. As shown in figure 4(b), the nodes (joints) in the tree only affect the position of their immediate children. The pelvis is the root of this tree; three branches lead from this root to nodes corresponding to the right hip, the left hip, and the lower spine⁷. Each hip joint is the parent node to a knee, and so on. We assign a conditional probability distribution, estimated from the corpus, to every (parent, child) pair in the tree. For every combination of

⁶Given this formulation, saying that two vertices are disconnected is synonymous with saying that two are connected by an edge with infinite weight.

⁷The sacrum and the five lumbar vertebrae are lumped together. This compromise sacrifices back suppleness for lowered complexity.

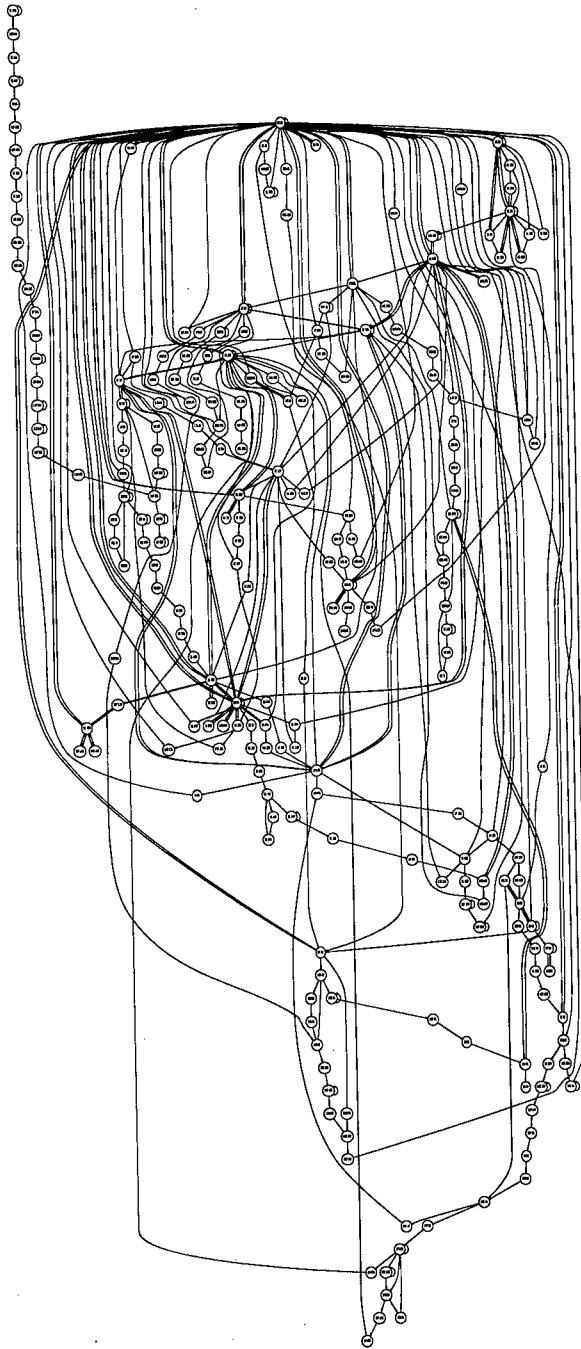


Figure 3: A transition graph that represents the movement patterns of the hips in a small corpus of 38 short ballet sequences. The numbers in each state identify the discretized position of the joint. Edge weights and isolated vertices have been omitted in the interests of clarity.

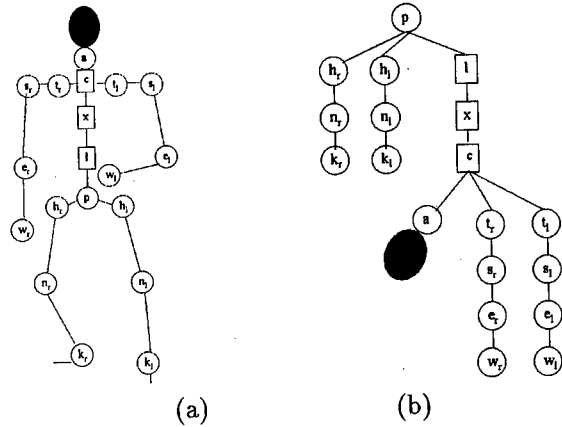


Figure 4: An influence diagram that explicitly represents the coordination of joints of the human body. Part (a) depicts the body and part (b) shows the inter-joint dependencies induced by gravity and topology: for instance, the position of the pelvis influences the positions of both hips h_r and h_l and the lumbar spine l , but the right and left ankles k_r and k_l do not directly influence one another. Without this simplifying assumption, the search space for this problem is intractable.

states that a parent λ and its child μ can assume, the distributions estimate the probability that joint μ is in orientation r given that joint λ is in orientation q , for every pair of discretized quaternions $q \in Q^\lambda, r \in Q^\mu$.

2.3 A JOINT-WISE INTERPOLATION ALGORITHM

Given a pair of discretized body postures (\vec{s}, \vec{t}) and a set of 23 transition graphs (one for each joint), we can use a memory-bounded A* search strategy (Winston 1992) to find an interpolation subsequence that moves smoothly between \vec{s} and \vec{t} . In general, A* finds a path from an initial state to a goal state by progressively generating successors of the current state in the search. The algorithm places successor states on a priority queue, sorted according to a score that estimates the cost of finding a goal state. In the next iteration, the state with the best score is drawn from the priority queue, its successor states are computed and added to the queue, and the procedure is repeated until a goal state is found or until the queue is empty.

In this application, the states in the A* search space are *body states* — 23-vectors of discretized quaternions that represent full body positions. To generate successors of a body state \vec{s} , we first use the transition graphs to find successors for each *joint state* s_λ independently, and then take all combinations (cross product) across the joints to obtain the list of body-

state successors. From this list, we can filter out the disallowed body positions using the influence diagram and the probability distribution of parent-child pairs. The successors of the joint-state s_λ are those vertices in G^λ that are connected to s_λ by an edge directed away from s_λ .

The score assigned to a body state \bar{u} has two parts:

1. the cost of the path from the initial state \bar{s} to \bar{u}
2. an estimate of the distance between \bar{u} and the goal state \bar{t}

The cost of the path starting at \bar{s} and ending at \bar{u} is simply the sum of the costs of the transitions taken in the path. Furthermore, since each body movement is composed of a group of joint movements, we can compute the cost of one body-state transition by summing the weights over the edges traversed by the joints. To make this concrete, suppose we are trying to find an interpolating path between the body states \bar{s} and \bar{t} . At some point in the search, we reach the body-state \bar{u} and must assign the path from \bar{s} to \bar{u} a score. If we write the path from \bar{s} to \bar{u} as $\bar{s} \rightsquigarrow \bar{u} = (\bar{x}^1 = \bar{s}, \bar{x}^2, \dots, \bar{x}^{z-1}, \bar{x}^z = \bar{u})$, we can express the cost of such a path as

$$g(\bar{s} \rightsquigarrow \bar{u}) = \sum_{i=1}^{z-1} \sum_{\lambda} w_{x_\lambda^i, x_\lambda^{i+1}}^\lambda$$

The heuristic part of the score, $h(\bar{u})$, estimates how far \bar{u} is from the goal state \bar{t} . $h(\bar{u})$ is calculated by summing the weights of the shortest paths from u_λ to t_λ , $u_\lambda, t_\lambda \in G^\lambda$ over all the joints. We obtain these shortest path weights using Dijkstra's single-source shortest path algorithm (Dijkstra 1959), implemented as described in (Cormen, Leiserson, & Rivest 1990). The final score assigned to body-state \bar{u} is then $f(\bar{s} \rightsquigarrow \bar{u}) = g(\bar{s} \rightsquigarrow \bar{u}) + h(\bar{u})$.

At the time of this writing, we have only done extensive testing on a greedy search strategy that ignores the cost of paths and scores nodes in the search based solely on the estimated distance between them and the goal (i.e., $f(\bar{s} \rightsquigarrow \bar{u}) = h(\bar{u})$). In the following section, we describe the implications of this strategy and suggest how different A* scoring functions are likely to affect the interpolation sequences. We are also working on incorporating more information about the position, velocity, and acceleration of the center of mass, so the momentum of the body is conserved as it passes through the interpolated sections of the movement; accomplishing this will require wide-ranging adaptations to the basic A* algorithm and perhaps even a wholly different approach. Finally, we are also in the process of testing how different influence diagram topologies affect the interpolation algorithm's ability to select good postures during the search. (For example, to

model and enforce the symmetry of the body, we could combine left and right counterparts into one node.)

3 RESULTS AND EVALUATION

The "goal" of choreography is aesthetic appeal, so it is difficult to analyze the results of this work using standard scientific methods⁸. However, there are some standard rules, procedures, and patterns in certain dance and martial arts genres that can be used to evaluate the interpolation sequences generated by the corpus-based techniques described in the previous sections. The evaluation described in this section is a highly condensed transcript of a dozen one-to two-hour sessions, wherein expert dancers — primarily Professor David Capps of the Department of Theater and Dance at the University of Colorado, an accomplished dancer and choreographer whose works have appeared on stages around the world, and Nadia Rojasadame, a student in that department and the composer of the adagio used to generate the variations shown in figures 1 and 2 — went through the results frame by frame, answering and then discussing the following questions:

- Does this posture transition look reasonable?
- If so, why and how?
- If not, why and how? What would you do instead? How many poses would you assume in doing so?

In order to make this process less subjective, we are developing a formal evaluation protocol, consisting of several subsequences and a series of scored questions about the flow of the movement therein, to be administered to groups of University of Colorado dance students.

Figure 5 shows a movement sequence that the learning and search algorithms described in the previous sections produced when given the task of interpolating between the fifth and sixth frames of the ballet sequence in figure 2. The search strategy was a simple greedy approach — an A* score $f(\bar{s} \rightsquigarrow \bar{u}) = h(\bar{u})$ that only factored in the distance to the goal — and the corpus included 38 short ballets. The starting and ending body postures (top left and top right in figure 5, labeled 1 and 10, respectively) are quite different; note the facing of the dancer and the weight distribution on the feet, for example. The eight-move interpolation sequence computed by the interpolator moves between those positions in a very natural way. Its first move, for instance, is to lower the left leg, a

⁸The very notion of objective, quantifiable evaluation elicited much consternation and mirth — along with some offense — from our expert dance consultants.

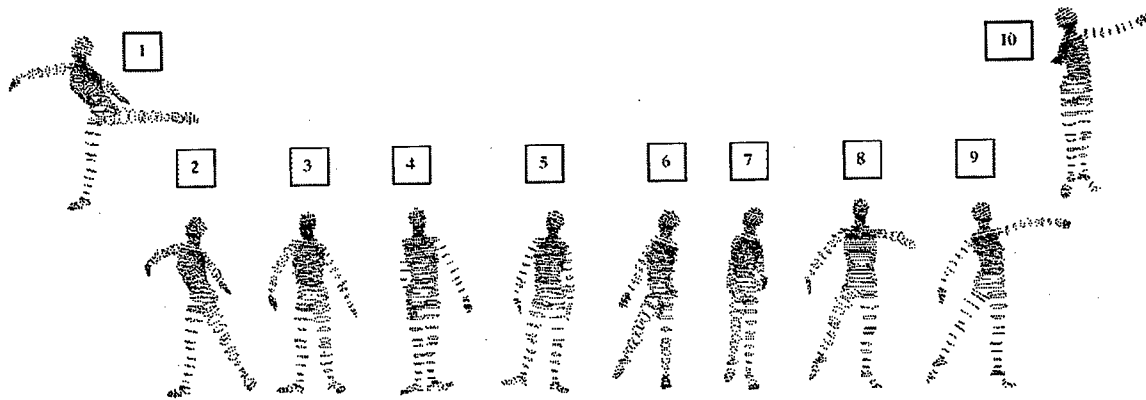


Figure 5: An interpolation sequence computed by the corpus-based techniques described in the previous section. The starting and ending positions passed as input to the interpolation procedure are shown at the top left and top right, respectively; the eight frames below them were computed by the interpolator.

natural strategy if one is going to change one's facing and end up on two feet. The following move is a simple weight shift (frames [4] and [5]), in preparation for a lift of the right leg. This lift, which is not strictly necessary to move from the fifth frame to the tenth, is an innovation that the program inserted because of the observed patterns in the corpus; it reflects the fact that ballet dancers rarely spin with *both* feet flat on the ground. Perhaps the most interesting thing about this interpolation sequence, from a balletic standpoint, is the *relevé*⁹ that the interpolation procedure inserted between frames [6] and [10]. Many *relevés* appear in the corpus, but none of them are associated with upper body positions that resemble the one that appears in this sequence. Our algorithm has invented a physically and stylistically appropriate way to move the dancer between the specified positions. The interpolation sequence in figure 5 includes a variety of other stylistically consistent innovations as well; consider, for example, the uplifted chest and chin in frames [7] and [9] — posture elements that are quintessential ballet style. Recall that these postures were not simply pasted in verbatim from the corpus; they were synthesized *joint by joint* using the transition graphs and influence-diagram directed A* search, and their fit to the genre is strong evidence of the success of the methods described in the previous section.

The original ballet sequence from which the snapshot in figure 1 was drawn contained 68 frames, and the chaotic shuffling scheme introduced 23 abrupt transitions into the variation (e.g., frames 5 ~ 6 of figure 2). In eleven of those 23 cases — including the one depicted in figure 5 — our interpolation scheme was successful in interpolating smoothly between the

two moves that framed the gap. The interpolation subsequences so constructed, which ranged in length from two to 60 frames, included a variety of stylistically consistent and often innovative sequences; among other things, the interpolation algorithms used *relevés*, *pliés* and fifth-position rests in highly appropriate ways — and all with no hard coding. From a subjective artistic standpoint, the results have some room for improvement; there are still five somewhat-awkward transitions in the 185 total frames of the 11 interpolation sequences. A less-subjective way to evaluate the success of this scheme is to compare the length of these interpolation sequences to the distance between the corresponding postures in the *original piece*, which is presumably a good metric for how long it would take a human to move from one to the other. For the most part, the interpolated sequences were shorter than or the same length¹⁰ as the number of frames separating the corresponding positions in the original piece, which indicates that the search strategies are working well. *mpeg* movies of this adagio sequence and its chaotic variation — both with and without interpolation — are available on the web¹¹.

This example brings out two significant failure modes of this approach. The algorithms cannot find interpolation subsequences between body positions that occur in reversed temporal order — e.g., places where the chaotic shuffler has forced a jump backwards in time, inserting a move into the variation that appeared earlier in the original piece. Secondly, the algorithms sometimes introduce relatively long paths between positions that appear very similar; in one such instance, where the task was a simple 90-degree rotation of the right shoulder around the long axis of the arm, the

⁹A *relevé*, which consists of lifting up on one's toes, is a stylistically required component of a direction shift in ballet.

¹⁰Five were shorter (77% average), four were the same length, and two were somewhat (150% and 110%) longer.

¹¹www.cs.colorado.edu/~lizb/chaotic-dance.html

algorithm constructed an 65-move sequence that involved much leg and trunk movement. Both of these problems are the result of limited corpus size and corresponding patterns in the joint transition graphs. These graphs are far from being connected, so some joint orientations are not reachable from others. Even when they are connected, the search may have to wander all over the graph to find a path between two given vertices. In a large, rich corpus, the graphs would be highly connected, giving the search algorithms more leeway. In the existing corpora, however, the paucity of edges constrains them to very narrow (and long) search paths that can translate to stilted, idiosyncratic movement sequences. This is an unavoidable problem in this application, unfortunately; the dance world has not yet embraced the notion of computer animation, so the availability of animated dances is quite limited.

Long, linear vertex chains like the ones at the top left of figure 3 are introduced into the joint transition graph when one animation in the corpus progresses through orientations that do not occur in other animations. The directionality in these chains makes it impossible for the search to move "upstream," which is the cause of the first failure mode described in the previous paragraph. We could fix this problem, artificially, by introducing reverse edges into the graphs in some kinesiology and stylistically justifiable way. For every transition $\vec{s} \rightarrow \vec{t}$ seen in the corpus, for example, we could introduce an edge from t_λ to \vec{s}_λ for every joint λ . The implicit assumption here is that *it is always possible to reverse the motion of a joint*¹². Thus, at the expense of destroying some of the accuracy with which the original approach modeled the temporal asymmetry of the genre, we could force the graphs to be connected. We are currently investigating what probabilities to use on these reverse edges. Artificially introduced reverse transitions would not solve the second problem; chains — even bidirectional chains — tend to lengthen interpolated sequences. One solution to this problem is to add more examples to the corpus to enrich its connectivity. If more examples are hard to come by, another (artificial) solution is to perform a coarser discretization to minimize the number of possible states a joint can assume. We are currently experimenting with different discretization resolutions to simultaneously minimize the number of nodes and maximize the statistical information content of the transition graphs.

The greedy A* search strategy is reflected by "inefficiencies" in the interpolation sequences — places where the dancer appears to be headed towards the goal state, but then moves away. For example, one

¹²This makes sense for classical ballet, but not modern dance; motion in the former tends to be "circular" in space, whereas in the latter, one often moves a limb out and back along the same path.

of the interpolation goals in figure 5 is to change the facing almost 180 degrees, from left to right. By the fourth frame, the dancer has turned to the right, but in the fifth frame s/he has turned back to the left again, which is part of what necessitates the *relevé* sequence between frames [6] and [7]. We are in the process of testing different search strategies and analyzing the results; instead of choosing the state that is closest to the goal, for instance, we are incorporating the path weights up to the current point in the solution as part of the scoring function. This should allow the search algorithm to find shorter, more-direct sequences. Finally, note that some search strategies — e.g., always taking the highest-probability branch — can be a significant source of cliché.

In order to explore the effects of joint coordination, we removed the influence diagram and ran simple, uncoordinated A* search to find paths between positions. The resulting sequences were extremely interesting. To the layman's eye, they look jerky and unappealing, so we expected negative comments about them from the experts. However, it seems that an uncoordinated path through a classical ballet corpus is a very good way to generate *modern* dance sequences, and the results were inventive and appealing: "Wow! I'm going to use *that* move in my next piece!" In retrospect, this makes some sense: the modern dance genre actively works at violating the ballet motif.

The interpolation procedure is fairly rapid. Applying greedy search to the 23 abrupt-transition pairs in the 68-frame variation, for instance, required¹³ 280 seconds on an HP9000/735 workstation running HP-UX v10.20 for a corpus containing 1720 ballet postures. A more-complex scoring function will obviously require longer run time. Preliminary runs of non-greedy A*, for example, required 500 seconds to perform the same task and yielded similar results, in terms of quality, sequence length, etc. The complexity also increases with corpus size; the same (non-greedy A*) task on an augmented corpus of 5000 postures — the 1720 original frames plus 3280 non-ballet sequences — required 3620 seconds. The chaotic shuffling procedure is also fast: for a 1000-position movement sequence, the chaotic shuffling procedure required 18 seconds on the same workstation, while a 9000-move sequence required 156 seconds.

4 CONCLUSION

By applying techniques from graph theory, artificial intelligence, and statistics to a corpus of movement sequences from a particular genre, the interpolation methods described in this paper automatically construct interpolation sequences that move from one

¹³This will obviously depend on the positions involved.

specified body posture to another in a *physically and stylistically coherent* fashion. These tactics can be used to smooth abrupt transitions that result from subsequence reordering, a common creative mechanism in modern choreography that can be emulated mathematically by using chaotic dynamics to generate variations.

Evaluating the results of this work is necessarily somewhat subjective. We have shown animations of a variety of different chaotic variations to hundreds of people, including dozens of dancers and martial artists, both with and without smoothing of the abrupt transitions. We have also worked in depth with several expert dancers in order to evaluate those interpolation sequences sensibly. The consensus is that the chaotic variations with smoothed transitions not only resemble the original pieces, but also are in some sense pleasing to the eye. They are both different from the originals and faithful to the dynamics of the genre; there are no jarring transitions or out-of-character moves. This is a non-trivial accomplishment. A previous attempt to use mathematics to generate choreographic variations — a subsequence randomization scheme introduced by the now well-known choreographer Merce Cunningham in the 1960s — met with a strongly negative reception in the dance world, *primarily because of the awkwardness at the transition points*¹⁴.

Many of the techniques used here, as well as others on which we are currently working, were inspired by solutions to similar problems that arise in computational linguistics (e.g., learning a grammar from a corpus and then using it to construct meaningful sentences). For example, one can view the transition graphs in section 2.2.1 and figure 3 as first-order Markov chains, where a single chain represents the probabilistic behavior of each joint in the body.

The objective of this research project was to tailor generic strategies for a specific high-dimensional search problem in an unusual and demanding domain. The results could be extended to other domains where the genre of sequence is important, such as speech recognition (e.g., filling in missing parts of a signal) or text. Finally, the implementation of these algorithms allows for arbitrary body topologies, so we are by no means limited to *human* motion sequences — though one would, of course, have to adapt the quaternion-based symbol set and the influence diagram to the topology of the limbs and joints that are involved.

Acknowledgements

The authors would like to thank D. Capps, N. Rojasadame, S. Schroeder, D. Jurafsky, M. Seltzer, D. Dabby, A. Hogan, E. Schell, A. Rubin, and the ICML-98 reviewers for helpful suggestions and comments. This work was supported by NSF NYI #CCR-9357740, ONR #N00014-96-1-0720, and a Packard Fellowship in Science and Engineering from the David and Lucile Packard Foundation.

References

- Bradley, E., and Stuart, J. 1997. Using chaos to generate choreographic variations. In *Proceedings of the Fourth Experimental Chaos Conference*.
- Bradley, E., and Stuart, J. 1998. Using chaos to generate variations on movement sequences. *Chaos*. To appear.
- Capps, D. 1998. University of Colorado, Department of Theater and Dance, personal communication.
- Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press. pp 527-531.
- Dabby, D. S. 1996. Musical variations from a chaotic mapping. *Chaos* 6:95-107.
- Dabby, D. S. 1997. A chaotic mapping for musical and image variation. In *Proceedings of the Fourth Experimental Chaos Conference*.
- Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269-271.
- Friedman, J. H.; Bentley, J. L.; and Finkel, R. A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software* 3:209-226.
- Goldstein, H. 1980. *Classical Mechanics*. Reading MA: Addison Wesley.
- Hodgins, J. K.; Wooten, W. L.; Brogan, D. C.; and O'Brien, J. F. 1995. Animating human athletics. In *Proceedings of SIGGRAPH*.
- Oliver, R. M., and Smith, J. Q., eds. 1990. *Influence Diagrams, Belief Nets and Decision Analysis*. Wiley.
- Winston, P. H. 1992. *Artificial Intelligence*. Redwood City CA: Addison Wesley. Third Edition.
- Wooten, W. L. 1998. *Simulation of Leaping, Tumbling, Landing, and Balancing*. Ph.D. Dissertation, Georgia Institute of Technology.

¹⁴Since that time, *aleatory* choreography — wherein randomization schemes are used to shuffle sequences — “has by now become one of the important currencies of dance composition approaches.” (Capps 1998).

Intra-Option Learning about Temporally Abstract Actions

Richard S. Sutton

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
rich@cs.umass.edu

Doina Precup

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
dprecup@cs.umass.edu

Satinder Singh

Department of Computer Science
University of Colorado
Boulder, CO 80309-0430
baveja@cs.colorado.edu

Abstract

Several researchers have proposed modeling temporally abstract actions in reinforcement learning by the combination of a policy and a termination condition, which we refer to as an *option*. Value functions over options and models of options can be learned using methods designed for semi-Markov decision processes (SMDPs). However, all these methods require an option to be executed to termination. In this paper we explore methods that learn about an option from small fragments of experience consistent with that option, even if the option itself is not executed. We call these methods *intra-option* learning methods because they learn from experience *within* an option. Intra-option methods are sometimes much more efficient than SMDP methods because they can use off-policy temporal-difference mechanisms to learn simultaneously about all the options consistent with an experience, not just the few that were actually executed. In this paper we present intra-option learning methods for learning value functions over options and for learning multi-time models of the consequences of options. We present computational examples in which these new methods learn much faster than SMDP methods and learn effectively when SMDP methods cannot learn at all. We also sketch a convergence proof for intra-option value learning.

these challenges within the framework of reinforcement learning and Markov decision processes (MDPs) (e.g., Singh, 1992a,b; Kaelbling, 1993; Lin, 1993; Dayan & Hinton, 1993; Thrun and Schwartz, 1995; Sutton, 1995; Huber and Grunewald, 1997; Kálmár, Szepesvári, and Lőrincz, 1997; Dietterich, 1998; Parr and Russell, 1998; Precup, Sutton, and Singh 1997, 1998a,b). This framework is appealing because of its general goal formulation, applicability to stochastic environments, and ability to use sample or simulation models (e.g., see Sutton and Barto, 1998). Extensions of MDPs to *semi-Markov decision processes* (SMDPs) provide a way to model temporally abstract actions, as we summarize in Sections 3 and 4 below. Common to much of this recent work is the modeling of a temporally extended action as a policy (controller) and a condition for terminating, which we together refer to as an *option*. Options are a flexible way of representing temporally extended courses of action such that they can be used interchangeably with primitive actions in existing learning and planning methods (Sutton, Precup, and Singh, in preparation).

In this paper we explore ways for learning about options using a class of off-policy, temporal-difference methods that we call *intra-option* learning methods. Intra-option methods look inside options to learn about them even when only a single action is taken that is consistent with them. Whereas SMDP methods treat options as indivisible black boxes, intra-option methods attempt to take advantage of their internal structure to speed learning. Intra-option methods were introduced by Sutton (1995), but only for a pure prediction case, with a single policy.

The structure of this paper is as follows. First we introduce the basic notation of reinforcement learning, options and models of options. In Section 4 we briefly review SMDP methods for learning value functions over options and thus how to select among options. Our new results are in Sections 5–7. Section 5 introduces an intra-option method for

1 Introduction

Learning, planning, and representing knowledge at multiple levels of temporal abstraction remain key challenges for AI. Recently, several researchers have begun to address

learning value functions and sketches a proof of its convergence. Computational experiments comparing it with SMDP methods are presented in Section 6. Section 7 concerns methods for learning *models* of options, as are used in planning: we introduce an intra-option method and illustrate its advantages in computational experiments.

2 Reinforcement Learning (MDP) Framework

In the reinforcement learning framework, a learning *agent* interacts with an *environment* at some discrete, lowest-level time scale $t = 0, 1, 2, \dots$. At each time step, the agent perceives the state of the environment, $s_t \in \mathcal{S}$, and on that basis chooses a primitive action, $a_t \in \mathcal{A}_{s_t}$. In response to a_t , the environment produces one step later a numerical reward, $r_{t+1} \in \mathbb{R}$, and a next state, s_{t+1} . We denote the union of the action sets by $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$. If \mathcal{S} and \mathcal{A} are finite, then the environment's transition dynamics are modeled by one-step state-transition probabilities, and one-step expected rewards,

$$\begin{aligned} p_{ss'}^a &= \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad \text{and} \\ r_s^a &= E\{r_{t+1} \mid s_t = s, a_t = a\}, \end{aligned}$$

for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ (it is understood here that $p_{ss'}^a = 0$ for $a \notin \mathcal{A}_s$). These two sets of quantities together constitute the *one-step model* of the environment.

The agent's objective is to learn a policy π , which is a mapping from states to probabilities of taking each action, that maximizes the expected discounted future reward from each state s :

$$V^\pi(s) = E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \mid s_t = s, \pi\},$$

where $\gamma \in [0, 1)$ is a *discount-rate* parameter. The quantity $V^\pi(s)$ is called the *value* of state s under policy π , and V^π is called the *value function* for policy π . The optimal value of a state is denoted

$$V^*(s) = \max_{\pi} V^\pi(s).$$

Particularly important for learning methods is a parallel set of value functions for state-action pairs rather than for states. The value of taking action a in state s under policy π , denoted $Q^\pi(s, a)$, is the expected discounted future reward starting in s , taking a , and henceforth following π :

$$Q^\pi(s, a) = E\{r_{t+1} + \gamma r_{t+2} + \dots \mid s_t = s, a_t = a, \pi\}.$$

This is known as the *action-value function* for policy π . The *optimal* action-value function is

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a).$$

The action value functions satisfy the Bellman equations:

$$Q^\pi(s, a) = r_s^a + \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s', a') Q^\pi(s', a') \quad (1)$$

$$Q^*(s, a) = r_s^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q^*(s', a'). \quad (2)$$

3 Options

We use the term *options* for our generalization of primitive actions to include temporally extended courses of action. In this paper, we focus on *Markov options*, which consist of three components: a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, a termination condition $\beta : \mathcal{S} \mapsto [0, 1]$, and an input set $\mathcal{I} \subseteq \mathcal{S}$. An option $(\mathcal{I}, \pi, \beta)$ is available in state s if and only if $s \in \mathcal{I}$. If the option is taken, then actions are selected according to π until the option terminates stochastically according to β . In particular, if the option taken in state s_t is Markov, then the next action a_t is selected according to the probability distribution $\pi(s_t, \cdot)$. The environment then makes a transition to state s_{t+1} , where the option either terminates, with probability $\beta(s_{t+1})$, or else continues, determining a_{t+1} according to $\pi(s_{t+1}, \cdot)$, possibly terminating in s_{t+2} according to $\beta(s_{t+2})$, and so on. When the option terminates, then the agent has the opportunity to select another option.

The input set and termination condition of an option together restrict its range of application in a potentially useful way. In particular, they limit the range over which the option's policy needs to be defined. For example, a hand-crafted policy π for a mobile robot to dock with its battery charger might be defined only for states \mathcal{I} in which the battery charger is within sight. The termination condition β would be defined to be 1 outside of \mathcal{I} and when the robot is successfully docked. For Markov options it is natural to assume that all states where an option might continue are also states where the option might be taken (i.e., that $\{s : \beta(s) < 1\} \subseteq \mathcal{I}$). In this case, π needs to be defined only over \mathcal{I} rather than over all of \mathcal{S} .

Given a set of options, their input sets implicitly define a set of available options \mathcal{O}_s for each state $s \in \mathcal{S}$. The sets \mathcal{O}_s are much like the sets of available actions, \mathcal{A}_s . We can unify these two kinds of sets by noting that actions can be considered a special case of options. Each action a corresponds to an option that is available whenever a is available ($\mathcal{I} = \{s : a \in \mathcal{A}_s\}$), that always lasts exactly one step ($\beta(s) = 1, \forall s \in \mathcal{S}$), and that selects a everywhere ($\pi(s, a) = 1, \forall s \in \mathcal{I}$). Thus, we can consider the agent's choice at each time to be entirely among options, some of which persist for a single time step, others which are more temporally extended. We refer to the former as *one-step* or *primitive* options and the latter as *multi-step* options.

We now consider Markov policies over options, $\mu : \mathcal{S} \times \mathcal{O} \mapsto [0, 1]$, and their value functions. When initiated in a state s_t , such a policy μ selects an option $o \in \mathcal{O}_{s_t}$ according to probability distribution $\mu(s_t, \cdot)$. The option o is taken in s_t , determining actions until it terminates in s_{t+k} , at which point a new option is selected, according to $\mu(s_{t+k}, \cdot)$, and so on. In this way a policy over options, μ , determines a policy over actions, or *flat policy*, $\pi = f(\mu)$. Henceforth we use the unqualified term *policy* for Markov policies over options, which include Markov flat policies as a special case.

Note, however, that $f(\mu)$ is typically not Markov because the action taken in a state depends on which option is being taken at the time, not just on the state. We define the value of a state s under a general flat policy π as the expected return if the policy is started in s :

$$V^\pi(s) \stackrel{\text{def}}{=} E\{r_{t+1} + \gamma r_{t+2} + \dots \mid \mathcal{E}(\pi, s, t)\},$$

where $\mathcal{E}(\pi, s, t)$ denotes the event of π being initiated in s at time t . The value of a state under a general policy (i.e., a policy over options) μ can then be defined as the value of the state under the corresponding flat policy: $V^\mu(s) \stackrel{\text{def}}{=} V^{f(\mu)}(s)$.

It is natural to also generalize the action-value function to an *option-value* function. We define $Q^\mu(s, o)$, the value of taking option o in state $s \in \mathcal{I}$ under policy μ , as

$$Q^\mu(s, o) \stackrel{\text{def}}{=} E\{r_{t+1} + \gamma r_{t+2} + \dots \mid \mathcal{E}(o\mu, s, t)\},$$

where $o\mu$, the *composition* of o and μ , denotes the policy that first follows o until it terminates and then initiates μ in the resultant state.

Options are closely related to the actions in a special kind of decision problem known as a *semi-Markov decision process*, or *SMDP* (e.g., see Puterman, 1994). Any fixed set of options for a given MDP defines a new SMDP overlaid on the MDP. The appropriate form of model for options, analogous to the r_s^a and $p_{ss'}^a$, defined earlier for actions, is known from existing SMDP theory. For each state in which an option may be started, this kind of model predicts the state in which the option will terminate and the total reward received along the way. These quantities are discounted in a particular way. For any option o , let $\mathcal{E}(o, s, t)$ denote the event of o being taken in state s at time t . Then the reward part of the model of o for state s is

$$r_s^o = E\{r_{t+1} + \gamma r_{t+2} \dots + \gamma^{k-1} r_{t+k} \mid \mathcal{E}(o, s, t)\}, \quad (3)$$

where $t+k$ is the random time at which o terminates. The

state-prediction part of the model of o for state s is

$$\begin{aligned} p_{ss'}^o &= \sum_{j=0}^{\infty} \gamma^j \Pr\{k = j, s_{t+j} = s' \mid \mathcal{E}(o, s, t)\} \\ &= E\{\gamma^k \delta_{s's_{t+k}} \mid \mathcal{E}(o, s, t)\}, \end{aligned} \quad (4)$$

for all $s' \in \mathcal{S}$, under the same conditions, where $\delta_{ss'}$ is an identity indicator, equal to 1 if $s = s'$, and equal to 0 else. Thus, $p_{ss'}^o$ is a combination of the likelihood that s' is the state in which o terminates together with a measure of how delayed that outcome is relative to γ . We call this kind of model a *multi-time model* because it describes the outcome of an option not at a single time but at potentially many different times, appropriately combined.

4 SMDP Learning Methods

Using multi-time models of options we can write Bellman equations for general policies and options. For example, the Bellman equation for the value of option o in state $s \in \mathcal{I}$ under a Markov policy μ is

$$Q^\mu(s, o) = r_s^o + \sum_{s'} p_{ss'}^o \sum_{o' \in \mathcal{O}_s} \mu(s', o') Q^\mu(s', o'). \quad (5)$$

The *optimal* value functions and *optimal* Bellman equations can also be generalized to options and to policies over options. Of course, the conventional optimal value functions V^* and Q^* are not affected by the introduction of options; one can ultimately do just as well with primitive actions as one can with options. Nevertheless, it is interesting to know how well one can do with a restricted set of options that does not include all the actions. For example, one might first consider only high-level options in order to find an approximate solution quickly. Let us denote the restricted set of options by \mathcal{O} and the set of all policies that select only from \mathcal{O} by $\Pi(\mathcal{O})$. Then the optimal value function given that we can select only from \mathcal{O} is

$$V_{\mathcal{O}}^*(s) \stackrel{\text{def}}{=} \max_{\mu \in \Pi(\mathcal{O})} V^\mu(s) \quad (6)$$

$$= \max_{o \in \mathcal{O}} E\{r + \gamma^k V_{\mathcal{O}}^*(s') \mid \mathcal{E}(o, s)\} \quad (7)$$

where $\mathcal{E}(o, s)$ denotes the event of starting the execution of option o in state s , k is the random number of steps elapsing during o , s' is the resulting next state, and r is the cumulative discounted reward received along the way. The optimal option values are defined as:

$$Q_{\mathcal{O}}^*(s, o) \stackrel{\text{def}}{=} \max_{\mu \in \Pi(\mathcal{O})} Q^\mu(s, o) \quad (8)$$

$$= E\{r + \gamma^k \max_{o' \in \mathcal{O}} Q_{\mathcal{O}}^*(s', o') \mid \mathcal{E}(o, s)\} \quad (9)$$

Given a set of options, \mathcal{O} , a corresponding *optimal policy*, denoted $\mu_{\mathcal{O}}^*$, is any policy that achieves $V_{\mathcal{O}}^*$, i.e., for which $V^{\mu_{\mathcal{O}}^*}(s) = V_{\mathcal{O}}^*(s)$ for all states $s \in \mathcal{S}$. If $V_{\mathcal{O}}^*$ and models of the options are known, then optimal policies can be formed by choosing in any proportion among the maximizing options in (7). Or, if $Q_{\mathcal{O}}^*$ is known, then optimal policies can be formed by choosing in each state s in any proportion among the options o for which $Q_{\mathcal{O}}^*(s, o) = \max_{o'} Q_{\mathcal{O}}^*(s, o')$. Thus, computing approximations to $V_{\mathcal{O}}^*$ or $Q_{\mathcal{O}}^*$ become the primary goals of planning and learning methods with options.

The problem of finding the optimal value functions for a set of options can be addressed by learning methods. Because an MDP augmented by options forms an SMDP, we can apply SMDP learning methods as developed by Bradtke and Duff (1995), Parr and Russell (1998), Parr (in preparation), Mahadevan et al. (1997), and McGovern, Sutton and Fagg (1997). In these methods, each option is viewed as an indivisible, opaque unit. After the execution of option o is started in state s , we next jump to the state s' in which it terminates. Based on this experience, an estimate $Q(s, o)$ of the optimal option-value function is updated. For example, the SMDP version of one-step Q-learning (Bradtke and Duff, 1995), which we call *one-step SMDP Q-learning*, updates after each option termination by

$$Q(s, o) \leftarrow Q(s, o) + \alpha \left[r + \gamma^k \max_{o' \in \mathcal{O}} Q(s', o') - Q(s, o) \right],$$

where k is the number of time steps elapsing between s and s' , r is the cumulative discounted reward over this time, and it is implicit that the step-size parameter α may depend arbitrarily on the states, option, and time steps. The estimate $Q(s, o)$ converges to $Q_{\mathcal{O}}^*(s, o)$ for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$ under conditions similar to those for conventional Q-learning (Parr, in preparation).

5 Intra-Option Value Learning

One drawback to SMDP learning methods is that they need to execute an option to termination before they can learn about it. Because of this, they can only be applied to one option at a time—the option that is executing at that time. More interesting and potentially more powerful methods are possible by taking advantage of the structure inside each option. In particular, if the options are Markov and we are willing to look *inside* them, then we can use special temporal-difference methods to learn usefully about an option before the option terminates. This is the main idea behind *intra-option* methods.

Intra-option methods are examples of *off-policy* learning methods (Sutton and Barto, 1998) in that they learn about

the consequences of one policy while actually behaving according to another, potentially different policy. Intra-option methods can be used to learn simultaneously about many different options from the same experience. Moreover, they can learn about the values of executing options *without ever executing* those options.

Intra-option methods for value learning are potentially more efficient than SMDP methods because they extract more training examples from the same experience. For example, suppose we are learning to approximate $Q_{\mathcal{O}}^*(s, o)$ and that o is Markov. Based on an execution of o from t to $t+k$, SMDP methods extract a single training example for $Q_{\mathcal{O}}^*(s, o)$. But because o is Markov, it is, in a sense, also initiated at each of the steps between t and $t+k$. The jumps from each intermediate s_{t+i} to s_{t+k} are also valid experiences with o , experiences that can be used to improve estimates of $Q_{\mathcal{O}}^*(s_{t+i}, o)$. Or consider an option that is very similar to o and which would have selected the same actions, but which would have terminated one step later, at $t+k+1$ rather than at $t+k$. Formally this is a different option, and formally it *was not executed*, yet all this experience could be used for learning relevant to it. In fact, an option can often learn something from experience that is only slightly related (occasionally selecting the same actions) to what would be generated by executing the option. This is the idea of off-policy training—to make full use of whatever experience occurs in order to learn as much possible about all options, irrespective of their role in generating the experience. To make the best use of experience we would like an off-policy and intra-option version of Q-learning.

It is convenient to introduce new notation for the value of a state–option pair given that the option is Markov and executing upon *arrival* in the state:

$$U_{\mathcal{O}}^*(s, o) = (1 - \beta(s))Q_{\mathcal{O}}^*(s, o) + \beta(s) \max_{o' \in \mathcal{O}} Q_{\mathcal{O}}^*(s, o'),$$

Then we can write Bellman-like equations that relate $Q_{\mathcal{O}}^*(s, o)$ to expected values of $\tilde{Q}_{\mathcal{O}}^*(s', o)$, where s' is the immediate successor to s after initiating Markov option $o = \langle \mathcal{I}, \pi, \beta \rangle$ in s :

$$\begin{aligned} Q_{\mathcal{O}}^*(s, o) &= \sum_{a \in \mathcal{A}_s} \pi(s, a) E \left\{ r + \gamma U_{\mathcal{O}}^*(s', o) \mid s, a \right\} \\ &= \sum_{a \in \mathcal{A}_s} \pi(s, a) \left[r_s^a + \sum_{s'} p_{ss'}^a U_{\mathcal{O}}^*(s', o) \right], \end{aligned}$$

where r is the immediate reward upon arrival in s' . Now consider learning methods based on this Bellman equation. Suppose action a_t is taken in state s_t to produce next state s_{t+1} and reward r_{t+1} , and that a_t was selected in a way consistent with the Markov policy π of an option

$o = \langle \mathcal{I}, \pi, \beta \rangle$. That is, suppose that a_t was selected according to the distribution $\pi(s_t, \cdot)$. Then the Bellman equation above suggests applying the off-policy one-step temporal-difference update:

$$Q(s_t, o) \leftarrow Q(s_t, o) + \alpha [r_{t+1} + \gamma U(s_{t+1}, o) - Q(s_t, o)],$$

where

$$U(s, o) = (1 - \beta(s))Q(s, o) + \beta(s) \max_{o' \in \mathcal{O}} Q(s, o')$$

The method we call *one-step intra-option Q-learning* applies this update rule to every option o consistent with every action taken a_t .

Theorem 1 (Convergence of intra-option Q-learning)

For any set of deterministic Markov options \mathcal{O} , one-step intra-option Q-learning converges w.p.1 to the optimal Q-values, $Q_{\mathcal{O}}^*$, for every option, regardless of what options are executed during learning, provided every primitive action gets executed in every state infinitely often.

Proof: (Sketch) On experiencing $\langle s, a, r, s' \rangle$, for every option o that picks action a in state s , intra-option Q-learning performs the following update:

$$Q(s, o) \leftarrow Q(s, o) + \alpha(s, o)[r + \gamma U(s', o) - Q(s, o)].$$

Let a be the action selection by deterministic Markov option $o = \langle \mathcal{I}, \pi, \beta \rangle$. Our result follows directly from Theorem 1 of Jaakkola et al. (1994) and the observation that the expected value of the update operator $r + \gamma U(s', o)$ yields a contraction, as shown below:

$$\begin{aligned} & |E\{r + \gamma U(s', o)\} - Q_{\mathcal{O}}^*(s, o)| \\ &= |r_s^a + \sum_{s'} p_{ss'}^a U(s', o) - Q_{\mathcal{O}}^*(s, o)| \\ &= |r_s^a + \sum_{s'} p_{ss'}^a U(s', o) - r_s^a - \sum_{s'} p_{ss'}^a U_{\mathcal{O}}^*(s', o)| \\ &\leq \left| \sum_{s'} p_{ss'}^a \left[(1 - \beta(s'))(Q(s', o) - Q_{\mathcal{O}}^*(s', o)) \right. \right. \\ &\quad \left. \left. + \beta(s')(\max_{o' \in \mathcal{O}} Q(s', o') - \max_{o' \in \mathcal{O}} Q_{\mathcal{O}}^*(s', o')) \right] \right| \\ &\leq \sum_{s'} p_{ss'}^a \max_{s'', o''} |Q(s'', o'') - Q_{\mathcal{O}}^*(s'', o'')| \\ &\leq \gamma \max_{s'', o''} |Q(s'', o'') - Q_{\mathcal{O}}^*(s'', o'')| \quad \diamond \end{aligned}$$

6 Illustrations of Intra-Option Value Learning

As an illustration of intra-option value-learning, we used the gridworld environment shown in Figure 1. The cells of

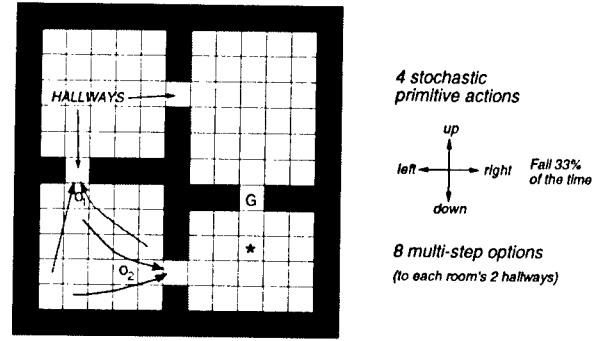


Figure 1: The rooms example is a gridworld environment with stochastic cell-to-cell actions and room-to-room hallway options. Two of the hallway options are suggested by the arrows labeled o_1 and o_2 . The label G indicates the location used as a goal.

the grid correspond to the states of the environment. From any state the agent can perform one of four actions, up, down, left or right, which have a stochastic effect. With probability $2/3$, the actions cause the agent to move one cell in the corresponding direction, and with probability $1/3$, the agent moves instead in one of the other three directions, each with $1/9$ probability. If the movement would take the agent into a wall, then the agent remains in the same cell. There are small negative rewards for each action, with means uniformly distributed between 0 and -1. The rewards are also perturbed by gaussian noise with standard deviation 0.1. The environment also has a goal state, labeled "G". A complete trip from a random start state to the goal state is called an *episode*. When the agent enters "G", it gets a reward of 1 and the episode ends. In all the experiments the discount parameter was $\gamma = 0.9$ and all the initial value estimates were 0.

In each of the four rooms we provide two built-in *hallway options* designed to take the agent from anywhere within the room to one of the two hallway cells leading out of the room. The policies underlying the options follow the shortest expected path to the hallway.

For the first experiment, we applied the intra-option method in this environment without selecting the hallway options. In each episode, the agent started at a random state in the environment and thereafter selected primitive actions randomly, with equal probability. On every transition, the update (5) was applied first to the primitive action taken, then to any of the hallway options that were consistent with it. The hallway options were updated in clockwise order, starting from any hallways that faced up from the current state. The value of the step-size parameter was $\alpha = 0.01$.

This is a case in which SMDP methods would not be able to

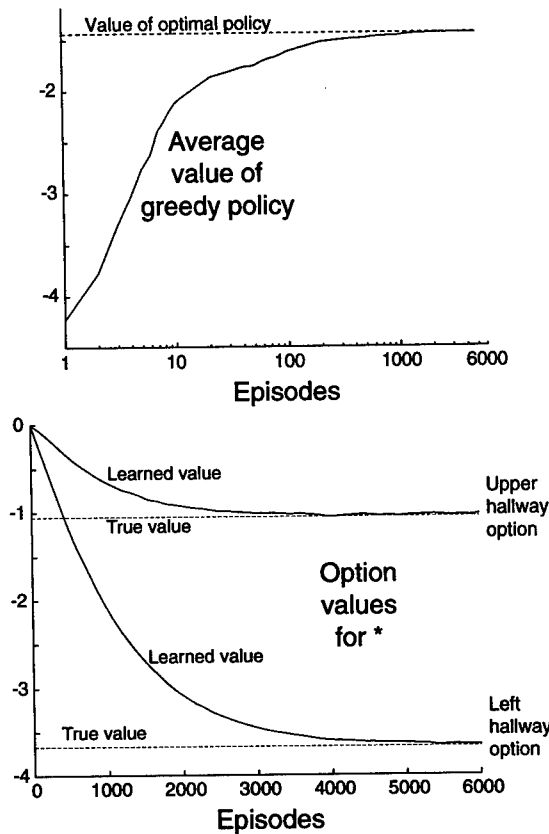


Figure 2: The learning of option values by intra-option methods without ever selecting the options. The value of the greedy policy goes to the optimal value (upper panel) as the learned values approach the correct values (as shown for one state, in the lower panel).

learn anything about the hallway options, because these options are never executed. However, the intra-option method learned the values of these actions effectively, as shown in Figure 2. The upper panel shows the value of the greedy policy learned by the intra-option method, averaged over \mathcal{I} and over 30 repetitions of the whole experiment. The lower panel shows the correct and learned values for the two hallway options that apply in the state marked * in Figure 1. Similar convergence to the true values was observed for all the other states and options.

So far we have illustrated the effectiveness of intra-option learning in a context in which SMDP methods do not apply. How do intra-option methods compare to SMDP methods when both are applicable? In order to investigate this question, we used the same environment, but now we allowed the agent to choose among the hallway options as well as the primitive actions, which were treated as one-step options. In this case, SMDP methods can be ap-

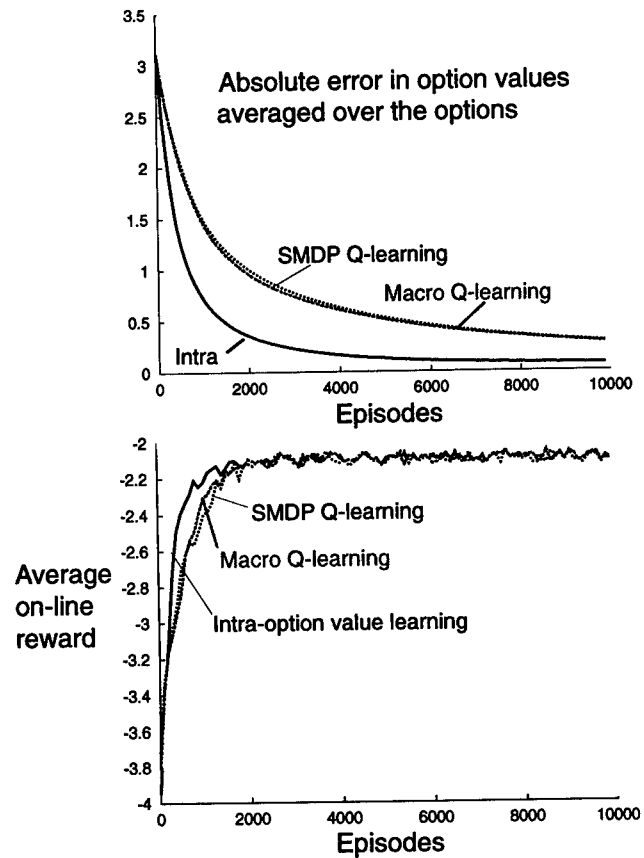


Figure 3: Comparison of SMDP, intra-option and macro Q-learning. Intra-option methods converge faster to the correct values.

plied, since all the options are actually executed. We experimented with two SMDP methods: one-step SMDP Q-learning (Bradtke and Duff, 1995) and a hierarchical form of Q-learning called *macro Q-learning* (McGovern, Sutton and Fagg, 1997). The difference between the two methods is that, when taking a multi-step option, SMDP Q-learning only updates the value of that option, whereas macro Q-learning also updates the values of the one-step options (actions) that were taken along the way.

In this experiment, options were selected not at random, but in an ϵ -greedy way dependent on the current option-value estimates. That is, given the current estimates $Q(s, o)$, let $o^* = \arg \max_{o \in \mathcal{O}_s} Q(s, o)$ denote the best valued action (with ties broken randomly). Then the policy used to select options was

$$\mu(s, o) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{O}_s|} & \text{if } o = o^* \\ \frac{\epsilon}{|\mathcal{O}_s|} & \text{otherwise,} \end{cases}$$

for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$. The probability of a random action, ϵ , was set at 0.1 in all cases. For each algorithm,

we tried step-size values of $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{8},$ and $\frac{1}{16}$ and then picked the best one.

Figure 3 shows two measures of the performance of the learning algorithms. The upper panel shows the average absolute error in the estimates of Q_O^* for the hallway options, averaged over the input sets \mathcal{I} , the eight hallway options, and 30 repetitions of the whole experiment. The intra-option method showed significantly faster learning than any of the SMDP methods. The lower panel shows the quality of the policy executed by each method, measured as the average reward over the state space. The intra-option method was also the fastest to learn by this measure.

7 Intra-Option Model Learning

In this section, we consider intra-option methods for learning multi-time models of options, r_s^o and $p_{ss'}^o$, given knowledge of the option (i.e., of its $\pi, \beta,$ and \mathcal{I}). Such models are used in planning methods (e.g., Precup, Sutton, and Singh, 1997, 1998a,b).

The most straightforward approach to learning the model of an option is to execute the option to termination many times in each state s , recording the resultant next states s' , cumulative discounted rewards r , and elapsed times k . These outcomes can then be averaged to approximate the expected values for r_s^o and $p_{ss'}^o$ given by (3) and (4). For example, an incremental learning rule for this could update its estimates \hat{r}_s^o and \hat{p}_{sx}^o , for all $x \in \mathcal{S}$, after each execution of o in state s , by

$$\hat{r}_s^o = \hat{r}_s^o + \alpha[r - \hat{r}_s^o], \quad \text{and} \quad (10)$$

$$\hat{p}_{sx}^o = \hat{p}_{sx}^o + \alpha[\gamma^k \delta_{sx} - \hat{p}_{sx}^o], \quad (11)$$

where the step-size parameter, α , may be constant or may depend on the state, option, and time. For example, if α is 1 divided by the number of times that o has been experienced in s , then these updates maintain the estimates as sample averages of the experienced outcomes. However the averaging is done, we call these *SMDP model-learning methods* because, like SMDP value-learning methods, they are based on jumping from initiation to termination of each option, ignoring what might happen along the way. In the special case in which o is a primitive action, note that SMDP model-learning methods reduce exactly to those used to learn conventional one-step models of actions.

Now let us consider intra-option methods for model learning. The idea is to use Bellman equations for the model, just as we used the Bellman equations in the case of learning value functions. The correct model of a Markov option

$o = \langle \mathcal{I}, \pi, \beta \rangle$ is related to itself by

$$r_s^o = \sum_{a \in \mathcal{A}_s} \pi(s, a) E \left\{ r + \gamma(1 - \beta(s')) r_{s'}^o \right\} \quad (12)$$

$$= \sum_{a \in \mathcal{A}_s} \pi(s, a) \left[r_s^a + \sum_{s'} p_{ss'}^a (1 - \beta(s')) r_{s'}^o \right] \quad (13)$$

where r and s' are the reward and next state given that action a is taken in state s , and

$$p_{sx}^o = \sum_{a \in \mathcal{A}_s} \pi(s, a) \gamma E \left\{ (1 - \beta(s')) p_{s'x}^o + \beta(s') \delta_{s'x} \right\} \\ = \sum_{a \in \mathcal{A}_s} \pi(s, a) \sum_{s'} p_{ss'}^a (1 - \beta(s')) p_{s'x}^o + \beta(s') \delta_{s'x}$$

for all $s, x \in \mathcal{S}$. How can we turn these Bellman equations into update rules for learning the model? First consider that action a_t is taken in s_t and that the way it was selected is consistent with $o = \langle \mathcal{I}, \pi, \beta \rangle$, that is, that a_t was selected with the distribution $\pi(s_t, \cdot)$. Then the Bellman equations above suggest the temporal-difference update rules

$$\hat{r}_{s_t}^o \leftarrow \hat{r}_{s_t}^o + \alpha \left[r_{t+1} + \gamma(1 - \beta(s_{t+1})) \hat{r}_{s_{t+1}}^o - \hat{r}_{s_t}^o \right] \quad (14)$$

and

$$\hat{p}_{s_t x}^o \leftarrow \hat{p}_{s_t x}^o + \alpha [\gamma(1 - \beta(s_{t+1})) \hat{p}_{s_{t+1} x}^o + \gamma \beta(s_{t+1}) \delta_{s_{t+1} x} - \hat{p}_{s_t x}^o], \quad (15)$$

where $\hat{p}_{ss'}^o$ and \hat{r}_s^o are the estimates of $p_{ss'}^o$ and r_s^o , respectively, and α is a positive step-size parameter. The method we call *one-step intra-option model learning* applies these updates to every option consistent with every action taken. Of course, this is just the simplest intra-option model-learning method. Others may be possible using eligibility traces and standard tricks for off-policy learning (see Sutton, 1995; Sutton and Barto, 1998).

Intra-option methods for model learning have advantages over SMDP methods similar to those we saw earlier for value-learning methods. As an illustration, consider the application of SMDP and intra-option model-learning methods to the rooms example. We assume that the eight hallway options are given as before, but now we assume that their models are not given and must be learned. Experience is generated by selecting randomly in each state among the two possible options and four possible actions, with no goal state. In the SMDP model-learning method, equations (10) and (11) were applied whenever an option was selected, whereas, in the intra-option model-learning method, equations (14) and (15) were applied on every step to all options that were consistent with the action taken on that step. In this example, all options are deterministic, so consistency

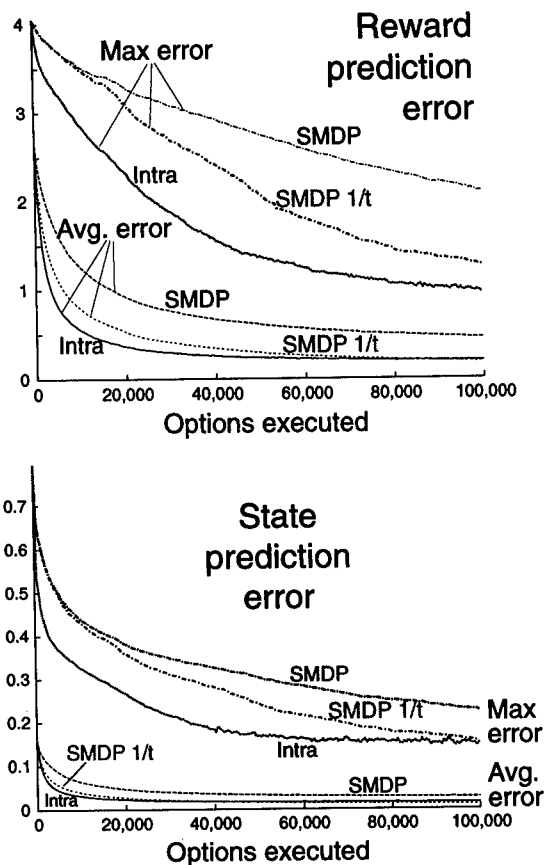


Figure 4: Learning curves for model learning by SMDP and intra-option methods.

with the action selected means simply that the option would have selected that action.

For the SMDP method, the step-size parameter was varied so that the model estimates were sample averages, which should give fastest learning. The results of this method are labeled "SMDP 1/t" on the graphs. We also looked at results using a fixed learning rate. In this case and for the intra-option method we tried step-size values of $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{8},$ and $\frac{1}{16}$, and picked the best value for each method. Figure 4 shows the learning curves for all three methods, using the best α values, when a fixed alpha was used. The upper panel shows the average and maximum absolute error in the reward predictions, and the lower panel shows the average absolute error and the maximum absolute error in the transition predictions, averaged over the eight options and over 30 independent runs. The intra-option method approached the correct values more rapidly than the SMDP methods.

8 Closing

The theoretical and empirical results presented in this paper suggest that intra-option methods provide an efficient way for taking advantage of the structure inside an option. Intra-option methods use experience with a single action to update the value or model for all the options that are consistent with that action. In this way they make much more efficient use of the experience than SMDP methods, which treat options as indivisible units. In the future, we plan to extend these algorithms for the case of non-Markov options, and to combine them with eligibility traces.

Acknowledgements

The authors acknowledge the help of their colleagues Amy McGovern, Andy Barto, Csaba Szepesvári, András Lörincz, Ron Parr, Tom Dietterich, Andrew Fagg, Leo Zelevinsky and Manfred Huber. We also thank Andy Barto, Paul Cohen, Robbie Moll, Mance Harmon, Sascha Engelbrecht, and Ted Perkins for helpful reactions and constructive criticism. This work was supported by NSF grant ECS-9511805 and grant AFOSR-F49620-96-1-0254, both to Andrew Barto and Richard Sutton. Doina Precup also acknowledges the support of the Fulbright foundation. Satinder Singh was supported by NSF grant IIS-9711753.

References

- Bradtke, S. J. & Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov decision problems. In *Advances in Neural Information Processing Systems 7* (pp. 393–400). MIT Press.
- Dayan, P. & Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5* (pp. 271–278). Morgan Kaufmann.
- Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann.
- Huber, M. & Grunen, R. A. (1997). A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, 22(3-4), 303–315.
- Jaakkola, T., Jordan, M. & Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), 1185–1201.
- Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the*

- Tenth International Conference on Machine Learning* (pp. 167–173). Morgan Kaufmann.
- Kalmár, Z., Szepesvári, C. & Lörincz, A. (1997). Module based reinforcement learning for a real robot. In *Proceedings of the Sixth European Workshop on Learning Robots* (pp. 22–32).
- Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University.
- Mahadevan, S., Marchallek, N., Das, T. K. & Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 202–210). Morgan Kaufmann.
- McGovern, A., Sutton, R. S. & Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. In *Grace Hopper Celebration of Women in Computing* (pp. 13–17).
- Parr, R. (in preparation). *Hierarchical Control and learning for Markov decision processes*. PhD thesis, Berkeley University. Chapter 3.
- Parr, R. & Russel, S. (1998). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*. MIT Press.
- Precup, D., Sutton, R. S. & Singh, S. (1997). Planning with closed-loop macro actions. In *Working Notes of the AAAI Fall Symposium '97 on Model-directed Autonomous Systems* (pp. 70–76).
- Precup, D., Sutton, R. S. & Singh, S. (1998a). Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems 10*. MIT Press.
- Precup, D., Sutton, R. S. & Singh, S. (1998b). Theoretical results on reinforcement learning with temporally abstract options. In *Machine Learning: ECML98. 10th European Conference on Machine Learning, Chemnitz, Germany, April 1998. Proceedings* (pp. 382–393). Springer Verlag.
- Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.
- Singh, S. P. (1992a). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 202–207). MIT/AAAI Press.
- Singh, S. P. (1992b). Scaling reinforcement learning by learning variable temporal resolution models. In *Proceedings of the Ninth International Conference on Machine Learning* (pp. 406–415). Morgan Kaufmann.
- Sutton, R. S. (1995). TD models: Modeling the world as a mixture of time scales. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 531–539). Morgan Kaufmann.
- Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.
- Sutton, R. S., Precup, D. & Singh, S. (in preparation). Between MDPs and Semi-MDPs: learning, planning, and representing knowledge at multiple temporal scales. *Journal of AI Research*.
- Thrun, S. & Schwartz, A. (1995). Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems 7* (pp. 385–392). MIT Press.

Teaching an Agent to Test Students

Gheorghe Tecuci

Computer Science Department
George Mason University
Fairfax, VA 22030-4444
tecuci@gmu.edu

Harry Keeling

Systems and Computer Science Department
Howard University
Washington D.C. 20059
hkeeling@scs.howard.edu

Abstract

This paper presents an innovative application of the Disciple Learning Agent Shell to the building of an educational agent that generates history tests for middle school students, to assist in the assessment of their understanding and use of higher-order thinking skills. Disciple has been taught by an educator to generate and answer basic test questions and to explain the answers. From its interaction with the educational expert, Disciple has learned general rules that allow it to generate a large number of new test questions for students, together with hints, answers, and explanations of the answers. As a result, it can guide the students during their practice of higher-order thinking skills as they would be directly guided by the educator. It can also be used by the educator to generate a different exam for each student in the class. Disciple has been experimentally evaluated by history experts, students and teachers, with very promising results. The work on developing this educational agent illustrates an integration of machine learning, knowledge acquisition, problem solving and intelligent tutoring systems in the context of computer-based assessment involving multimedia documents.

1 INTRODUCTION

For several years we have been developing the Disciple approach for building intelligent agents. The defining feature of the Disciple approach to building agents is that a person teaches the agent how to perform domain-specific tasks, by giving the agent examples and explanations, as well as supervising and correcting its behavior. The current version of the Disciple approach is implemented in the Disciple Learning Agent Shell, and is presented in (Tecuci, 1998). We define a *learning agent shell* as consisting of a learning engine and an inference engine that support a representation formalism in which a knowledge base can be encoded, as well as a methodology for building the knowledge base.

The central goal of the Disciple approach is to facilitate the agent building process by the use of synergism at

three different levels. First, there is synergism between different learning methods employed by the agent (Michalski and Tecuci, 1994). By integrating complementary learning methods (such as inductive learning from examples, explanation-based learning, learning by analogy, learning by experimentation), the Disciple agent is able to learn from the human expert in situations in which no single strategy learning method would be sufficient. Second, there is synergism between expert's teaching of the agent and the agent's learning from the expert (Tecuci and Kodratoff, 1995). For instance, the expert may select representative examples to teach the agent, may provide explanations, and may answer agent's questions. The agent, on the other hand, will learn general rules that are difficult to be defined by the expert, and will consistently integrate them into its knowledge base. Third, there is synergism between the expert and the agent in solving a problem. They form a team in which the agent solves the more routine but labor intensive parts of the problem and the expert solves the more creative parts. In the process, the agent learns from the expert, gradually evolving toward an "intelligent" agent (Mitchell et al., 1985). We claim that the Disciple approach significantly reduces the involvement of the knowledge engineer in the process of building an intelligent agent, most of the work being done directly by the domain expert. In this respect, the work on Disciple is part of a long term vision where personal computer users will no longer be simply consumers of ready-made software, as they are today, but also developers of their own software assistants.

This paper is organized as follows. The next section presents the developed test generation agent. Then, sections 3, 4 and 5 describe the process of building the agent. Section 6 describes the results of the experiments performed with the developed agent. Finally, the paper presents the conclusions of this work.

2 A TEST GENERATION AGENT

We have developed an agent that generates history tests to assist in the assessment of students' understanding and use of higher-order thinking skills. Examples of specific higher-order thinking skills are: *evaluation* of historical sources for relevance, credibility, consistency, ambiguity,

bias, and fact vs. opinion; *analyzing* them for content, meaning and point of view; and *synthesizing* arguments in the form of conclusions, claims and assertions (Bloom, 1956; Beyer, 1987, 1988).

To motivate the middle school students, for which this agent was developed, and to provide an element of game playing, the agent employs a journalist metaphor, asking the students to assume the role of a novice journalist. Figure 1, for instance, shows a test question generated by the agent. The student is asked to imagine that he or she is a reporter and has been assigned the task to write an article for *Christian Recorder* during the Civil War period on plantations. The student has to analyze the historical source "Slave Quarters" in order to determine whether it is relevant to this task. In the situation illustrated in Figure 1 the student answered correctly. Therefore, the agent confirmed the answer and provided an explanation for it, as indicated in the lower right pane of the window. The student could have requested a hint to answer the question and would have received the following one: "To determine if the source is relevant to your task investigate if it illustrates some component of a plantation, check when it was created and when *Christian Recorder* was issued." In general, there may be several reasons why a source is relevant to a task. By pushing the More button, the student can receive the hints and explanations corresponding to these additional reasons.

Another example of a test question is shown in Figure 2. The student is given a task, a historical source and three possible reasons why the source is relevant to the task. He or she has to investigate the source and decide which reason(s) account for the fact that the source is relevant to the task. The student is instructed to check the box next to the correct reason(s).

The agent has two modes of operation: final exam mode and self-assessment mode. In the final exam mode, the agent generates an exam consisting of a set of test questions of different levels of difficulty. The student has to answer one test question at a time and, after each question, he or she receives the correct answer and an explanation of the answer. In the self-assessment mode, the student chooses the type of test question to solve, and will receive, on request, feedback in the form of hints to answer the question, the correct answer, and some or all the explanations of the answer. The test questions are generated such that all students interacting with the agent are likely to receive different tests even if they follow exactly the same interaction pattern. Moreover, the agent builds and maintains a simple student model and uses it in the process of test generation. For instance, to the extent possible, the agent tries to generate test questions that involve historical sources that have not been investigated by the student, or historical sources that were not used in previous tests for that student.

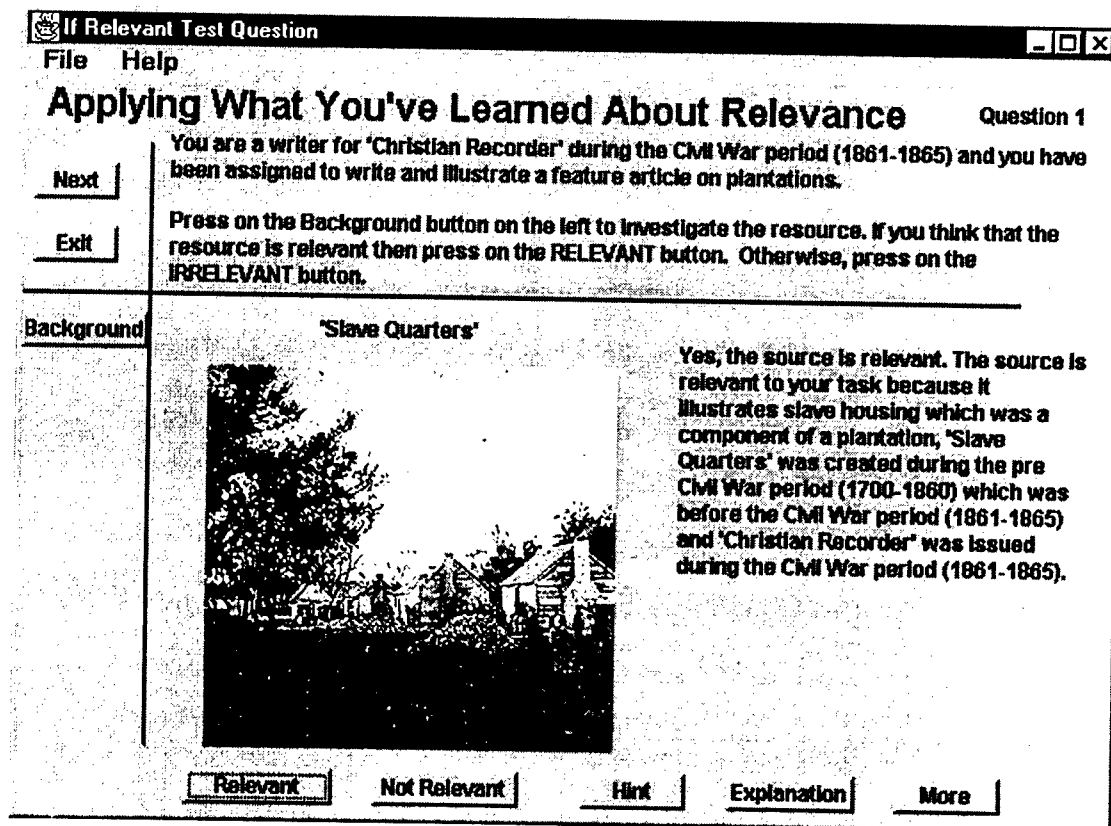
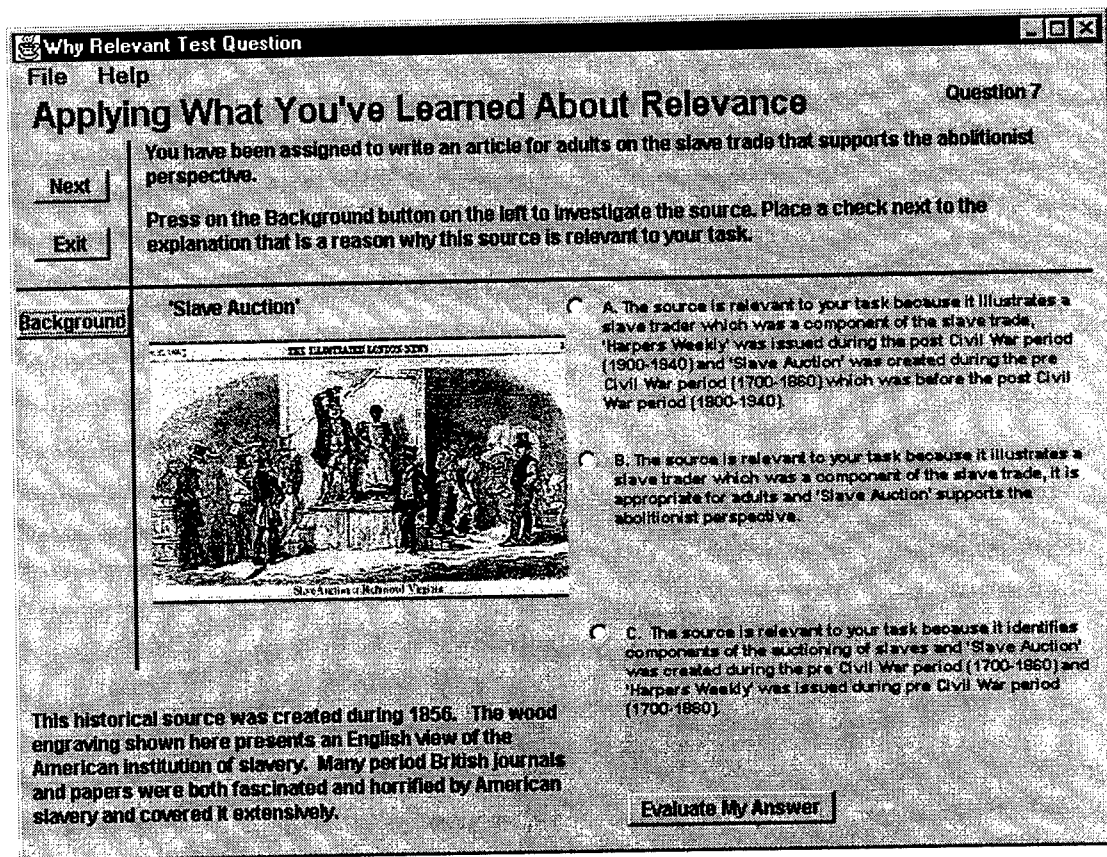


Figure 1: A test question, answer and explanation generated by the agent¹

¹ Picture reproduced from LC-USZ62-67818, Library of Congress, Prints & Photographs Division, Civil War Photographs

Figure 2: Another test question²

The next sections present the process of building this agent: building the agent's ontology (Gruber, 1993), teaching the agent how to generate test questions, and building the test generation engine.

3 BUILDING THE AGENT'S ONTOLOGY

The agent's ontology contains descriptions of historical concepts (such as "plantation"), historical sources (such as "Slave Quarters" in Figure 1), and templates for reporter tasks (such as "You are a writer for PUBLICATION during HISTORICAL-PERIOD and you have been assigned to write and illustrate a feature article on SLAVERY-TOPIC."). Using these descriptions and templates, the agent communicates with the students through a stylized natural language, as illustrated in Figure 1 and Figure 2.

The ontology building process starts with choosing a module in a history curriculum (such as Slavery in America) for which the agent will generate test questions. Then the educator identifies a set of historical concepts that are appropriate and necessary to be learned by the students. The educator also identifies a set of historical sources that can enhance the student's understanding of these concepts and will be used in test questions. All these concepts and historical sources are represented by

the history educator in the knowledge base, by using the various interfaces of Disciple. One is the Source Viewer that displays the historical sources. Another is the Concept Editor that is used to describe the historical sources. The historical sources have to be defined in terms of features that are necessary for applying the higher-order thinking skills of relevance, credibility, etc. For instance, a source is relevant to some topic if it identifies, illustrates or explains the topic or some of its components. Let us consider the historical source 'Contented Slaves and Masters', from the bottom of Figure 3. This source is defined as being a LITHOGRAPH that ILLUSTRATES the concepts SLAVE-DANCE, MALE-SLAVE, FEMALE-SLAVE, and SLAVE-MASTER. Other information has also to be represented, such as the audience for which this source is appropriate and when it was created. The concepts from the knowledge base are hierarchically organized in a semantic network (Quillian, 1968; Lenat and Guha, 1990) that can be inspected with the Concept Browser. For instance, SLAVE-DANCE was defined as being a type of SLAVE-RECREATION which, in turn, was a SLAVE-LIFE-ASPECT. This initial knowledge base of the agent was assumed to be incomplete and even possibly partially incorrect, needing to be improved during the next stages of the agent's development.

² Picture reproduced from LC-USZ62-15398, Library of Congress, Prints & Photographs Division, Civil War Photographs

4 TEACHING THE AGENT

A basic relevancy test question consists of judging the relevancy of a historical source to a given reporter's task. To teach the agent to generate and answer such questions, the educator gives it an example consisting of a task and a historical source relevant to that task, as shown in Figure 3. Starting from this example, the agent has learned the relevancy rule in Figure 4, where the condition specifies a general reporter task and the conclusion specifies a source relevant to that task. The condition also incorporates the explanation of why the source is relevant to the task. Associated with the rule are the natural language templates corresponding to its task, explanation and conclusion. They are automatically created from the natural language descriptions of the elements in the rule. One should notice that each rule corresponds to a certain type of task (WRITE-DURING-PERIOD, in this case). Other types of tasks are WRITE-ON-TOPIC, WRITE-FOR-AUDIENCE, and WRITE-FOR-OCCASION. Therefore, for each type of reporter task there will be a family of related relevancy rules. The rules corresponding to the other evaluation criteria, such as credibility, accuracy, or bias, will have a similar form.

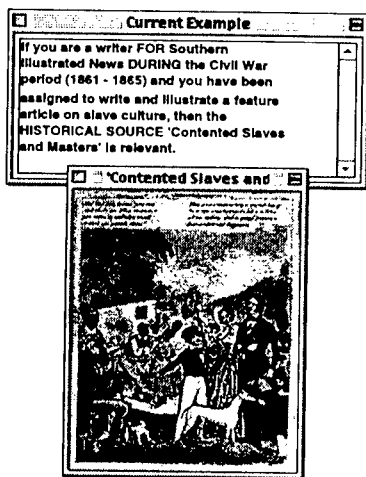


Figure 3: Initial example given by the educator³

IF	
?W1 IS	WRITE-DURING-PERIOD, FOR ?S1, DURING ?P1, ON ?S2
?S1 IS	PUBLICATION, ISSUED-DURING ?P1
?P1 IS	HISTORICAL-PERIOD
?S2 IS	SLAVERY-TOPIC
?S3 IS	SOURCE, ILLUSTRATES ?S4, CREATED-DURING ?P2
?S4 IS	HISTORICAL-CONCEPT, COMPONENT-OF ?S2
?P2 IS	HISTORICAL-PERIOD, BEFORE ?P1
THEN	
RELEVANT HIST-SOURCE ?S3	
Task Description: You are a writer for ?S1 during ?P1 and you have been assigned to write and illustrate a feature article on ?S2	
Explanation: ?S3 illustrates ?S4 which was a component of ?S2, ?S3 was created during ?P2 which was before ?P1 and ?S1 was issued during ?P1.	
Operation Description: ?S3 is relevant	

Figure 4: A relevancy rule

4.1 RULE LEARNING

The rule learning method of Disciple is schematically represented in Figure 5. As Explanation-based Learning (DeJong and Mooney, 1986; Mitchell, Keller, Kedar-Cabelli, 1986), it consists of two phases, explanation and generalization. However, in the explanation phase the agent is not building a proof tree, and the generalization is not a deductive one.

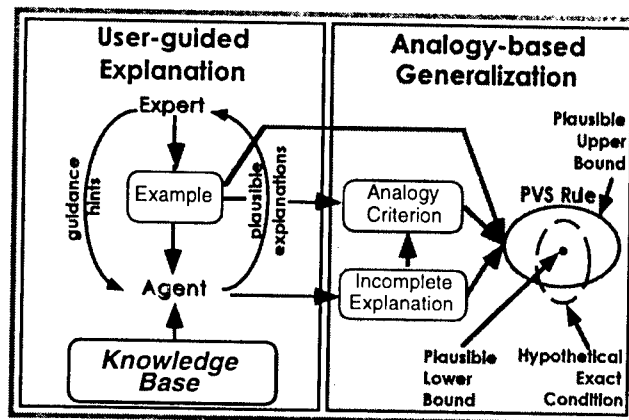


Figure 5: The rule learning method of Disciple

In the explanation phase, the educator helps the agent to understand why the example in Figure 3 is correct (that is, why the source is relevant to the given task). The explanation of the example has a form that is similar to the one given by a teacher to a student: the source "Contented Slaves and Masters" is relevant to the given task (see Figure 3) because it illustrates a slave dance which was a component of slave culture, and it was created during the pre Civil War period which was before the Civil War period. Each of these phrases corresponds to a path in the agent's ontology, as shown in Figure 6. However, rather than giving an explanation to the agent, the educator guides it to propose explanations and then selects the correct ones. For instance, the educator may point to the most relevant objects from the input example and may specify the types of explanations to be generated by the agent (e.g. a correlation between two objects or a property of an object). The agent uses such guidance and specific heuristics to propose plausible explanations to the educator who has to select the correct ones. A particularly useful heuristic is to propose explanations of an example by analogy with the explanations of other examples. Notice that the above explanation is similar to a part of the explanation from the test question in Figure 1. This illustrates a significant benefit to be derived from using the Disciple approach to build educational agents. That is, the kind of explanations that the agent gives to the students are similar to the explanations that the agent itself has received from the educator. Therefore, the agent acts as an indirect communication medium between the educator and the students.

In the generalization phase (see Figure 5), the agent performs an analogy-based generalization of the example

³ Picture reproduced from LC-USZ62-89745, Library of Congress

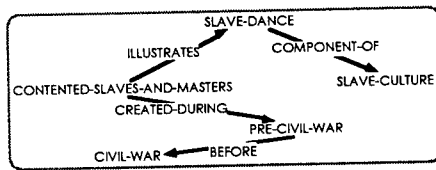


Figure 6: The explanation of the example in Figure 3

and its explanation into a plausible version space (PVS) rule. A PVS rule is an IF-THEN rule with two conditions, a plausible upper bound condition that is likely to be more general than the exact condition, and a plausible lower bound condition that is likely to be less general than the exact condition. The generalization process is illustrated in Figure 7. The initial example is the internal representation of the example in Figure 3. Also, the explanation is the one from Figure 6. First, the explanation is generalized to an analogy criterion by preserving the object features (such as ILLUSTRATES and CREATED-DURING) and by generalizing the objects to more general concepts (e.g. generalizing SLAVE-DANCE to HISTORICAL-CONCEPT). To determine how to generalize an object, Disciple analyzes all the features from the example and the explanation that are connected to that object. Each such feature is defined in Disciple's ontology by a domain (that specifies the set of all the objects from the application domain that may have that feature) and a range (that specifies all the possible values of that feature). The domains and the ranges of these features restrict the generalizations of the objects. For instance, in the explanation from Figure 7, SLAVE-DANCE has the feature COMPONENT-OF and appears as value of the feature ILLUSTRATES. Therefore, the most general generalization of SLAVE-DANCE is the intersection of the domain of COMPONENT-OF and the range of ILLUSTRATES, which is HISTORICAL-CONCEPT.

The analogy criterion and the example are used to generate the plausible upper bound condition of the rule, while the explanation and the example are used to generate the plausible lower bound condition of the rule.

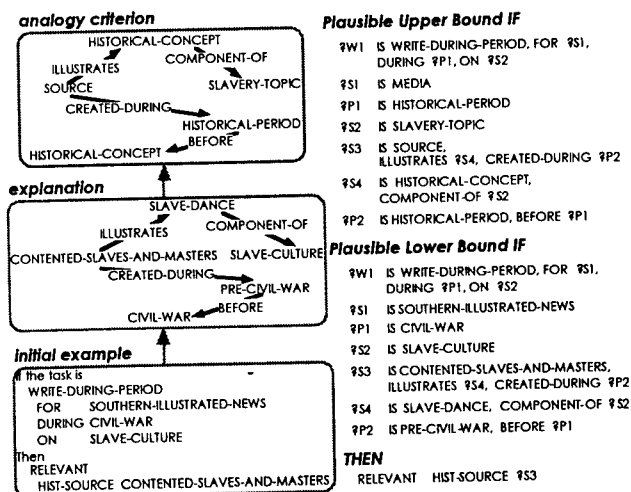


Figure 7: Generation of initial plausible version space rule

4.2 RULE REFINEMENT

The representation of the PVS rule in the right hand side of Figure 5 shows the most likely relation between the plausible lower bound, the plausible upper bound and the hypothetical exact condition of the rule. Notice that there are instances of the plausible upper bound that are not instances of the hypothetical exact condition of the rule. This means that the learned rule in Figure 7 covers also some negative examples. Also, there are instances of the hypothetical exact condition that are not instances of the plausible upper bound. This means that the plausible upper bound does not cover all the positive examples of the rule. Both of these situations are a consequence of the fact that the explanation of the initial example might be incomplete, and are consistent with what one would expect from an agent performing analogical reasoning. To improve this rule, the agent will use the rule refinement method represented schematically in Figure 8. The agent will use the learned rule to generate examples similar with the one in Figure 3. Each such example is covered by the plausible upper bound and is not covered by the plausible lower bound of the rule. The example is shown to the educator who is asked to accept it as correct or to reject it, thus characterizing it as a positive or a negative example of the rule. A correct example is used to generalize the plausible lower bound of the rule's condition through empirical induction. An incorrect example is used to elicit additional explanations from the educator and to specialize both bounds, or only the upper bound.

Figure 9 shows an example generated by the agent, by analogy with the initial example in Figure 3. The agent's analogical reasoning is represented in Figure 10. The explanation from the left hand side indicates why the initial example is correct. The expression from its right side is similar with this explanation because both of them are less general than the analogy criterion from the top of Figure 10. Therefore, one may infer by analogy that the similar explanation from the right hand side of Figure 10 explains an example (the generated example from the right hand side of Figure 10 and from Figure 9) that is similar to the initial example. Nevertheless, the generated example is incorrect and was rejected by the educator.

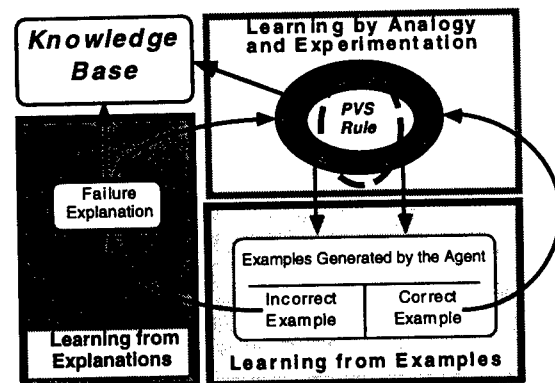
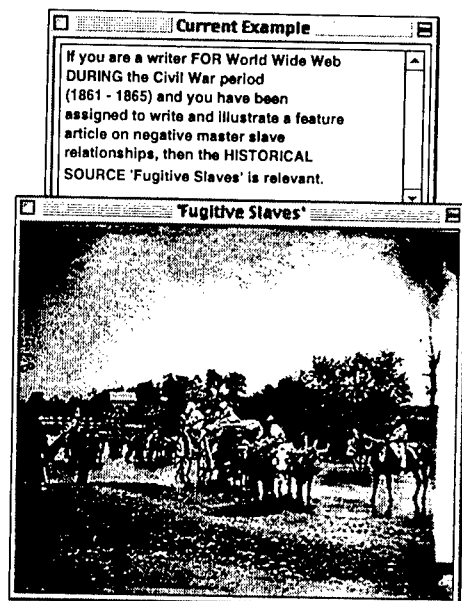


Figure 8: The rule refinement method of Disciple

Figure 9: An example generated by the agent⁴

In such a case the agent needed to understand why this example, which was generated by analogy with a correct example, is wrong. By comparing the two examples, the educator and the agent were able to find out that the generated example is wrong because the WORLD-WIDE-WEB was not issued during the CIVIL-WAR period. On the contrary, the initial example was correct because SOUTHERN-ILLUSTRATED-NEWS was issued during the CIVIL-WAR period. This explanation is used to specialize both bounds of the version space. This process will continue until either the two bounds of the rule become identical or until no further examples can be generated that are not already covered by the plausible lower bound. The final rule is the one from Figure 4. This training phase continued until 54 relevancy rules were learned.

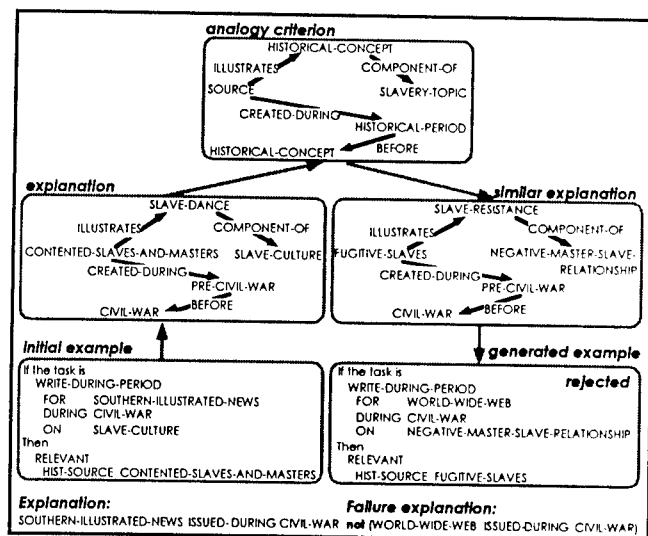


Figure 10: Analogical reasoning in Disciple

5 THE TEST GENERATION ENGINE

One of the agent's requirements was that it generates not only test questions, but also feedback for right and wrong answers, hints to help the student in solving the tests, as well as explanations of the solutions. Moreover, agent's messages needed to be expressed in a natural language form. Although the rules learned by the agent contain almost all the necessary information to achieve these goals, some small adjustments were necessary. In the case of the rule in Figure 4, the educator needed to define the templates for the Hint, Right Answer and Wrong Answer, shown in Figure 11. The Hint in Figure 11 is the part of the Explanation in Figure 4 that refers only to the variables used in the formulation of the test question. The Right Answer in Figure 11 is generated from the Operation Description and the Explanation in Figure 4, and the Wrong Answer is a fixed text.

Hint: To determine if this source is relevant to your task investigate if it illustrates some component of ?S2, check when was it created, and when ?S1 was issued

Right Answer: The source ?S3 is relevant to your task because it illustrates ?S4 which was a component of ?S2, ?S3 was created during ?P2 which was before ?P1 and ?S1 was issued during ?P1.

Wrong Answer: Investigate this source further and analyze the hints and explanations to improve your understanding of relevance. You may consider reviewing the material on relevance. Then continue testing yourself.

Figure 11: Additional templates for the rule in Figure 4

The learned rules can be used to generate different types of tests. In the current version of the agent we have chosen to develop a test generation engine that can generate the following four classes of test questions:

- **IF RELEVANT:** Show the student a writing assignment and ask whether a particular historical source is relevant to that assignment;
- **WHICH RELEVANT:** Show the student a writing assignment and three historical sources and ask the student to identify the relevant one;
- **WHICH IRRELEVANT:** Show the student a writing assignment and three historical sources and ask the student to identify the irrelevant one; and
- **WHY RELEVANT:** Show the student a writing assignment, a source and three possible reasons why the source is relevant, and ask the student to select the right reason.

Similar questions could be generated for other evaluation skill such as IF CREDIBLE or WHY CREDIBLE test questions.

To generate an IF RELEVANT test question with a relevant source, the agent simply needs to generate an example of a relevancy rule. This rule example will contain a task T and a source S relevant to it, together with one hint and one explanation that will indicate one reason why S is relevant to T. However, if the student requires all the possible reasons for why the source S is relevant to T, then the agent will need to find all the examples containing the source S and the task T of all the relevancy rules from the family of rules corresponding to T.

⁴ Picture reproduced from LC-USZ622-14828, Library of Congress

To generate an IF RELEVANT test question with an irrelevant source, the agent has first to generate a valid task T by finding an example of a relevancy rule R. Then it has to find a historical source S such that the task T and the source S are not part of an example of any rule from the family of rules corresponding to the task T.

The methods for generating WHICH RELEVANT and WHICH IRRELEVANT test questions are based on the methods for generating IF RELEVANT test questions.

For an WHY RELEVANT test question an example E_1 of a relevancy rule R_1 is generated. This example provides a correct task description T, a source S relevant to T, and a correct explanation EX_1 of why the source S is relevant to T. Then the agent chooses another rule that is not from the family of the relevancy rules corresponding to T. This rule could be from another family of relevancy rules, or could be a rule corresponding to another evaluation skill, for instance credibility or accuracy. Let us suppose that the agent chooses a credibility rule R_2 . It then generates an example E_2 of R_2 , based on E_1 (that is, E_2 and E_1 share as many parts as possible, including the source S). The agent also generates an explanation EX_2 of why S is credible. While this explanation is correct, it has nothing to do with why S is relevant to T. Then, the agent repeats this process to find another explanation that is true but explains something else, not why S is relevant to T.

It should be noticed that, when the agent has to choose an element from a set, the choice is done at random. Thus, its behavior is different from one execution to another.

6 EXPERIMENTAL RESULTS

The ontology of the test generation agent includes the description of 252 historical concepts, 80 historical sources, and 6 publications. The knowledge base also contains 54 relevancy rules grouped in four families, each family corresponding to one type of reporter task. These rules have been learned from an average of 2.17 explanations (standard deviation 0.91) and 5.4 examples (standard deviation 1.37).

There are 40,930 instances of the 54 relevancy rules in the knowledge base. Each such instance corresponds to an IF RELEVANT test question where the source is relevant. In

principle, for each such test question the agent can generate several IF RELEVANT test questions where the source is not relevant, as well as several WHY RELEVANT, WHICH RELEVANT and WHICH IRRELEVANT test questions. Therefore, the agent can generate more than 10^5 different test questions.

We have performed four types of experiments with the test generation agent. The first experiment tested the correctness of the knowledge base, as judged by the domain expert who developed the agent. This was intended to clarify how well the developed agent represents the expertise of the teaching expert. The second experiment tested the correctness of the knowledge base, as judged by a domain expert who was not involved in its development. This was intended to test the generality of the agent, given that assessing relevance is, to a certain extent, a subjective judgment. The third and the fourth experiments tested the quality of the test generation agent, as judged by students and by teachers.

The results of the first two experiments are summarized in Table 1. To test the predictive accuracy of the knowledge base, 406 IF RELEVANT test questions were randomly generated by the agent and answered by the developing expert. We have performed a similar experiment with a domain expert who was not involved in the development of the agent. This independent expert has answered another 401 randomly generated IF RELEVANT test questions. These experiments have revealed a much higher predictive accuracy in the case of IF RELEVANT test questions where the source was relevant. This was 96.53% in the case of the developing expert and 95.45% in the case of the independent expert. The predictive accuracy in the case of irrelevant sources was only 81.86% in the case of the developing expert and 76.35% in the case of the independent expert. To confirm these results we have conducted an additional experiment with the independent expert, who was shown other 1,326 IF RELEVANT test questions where all the sources were relevant (for a total of 1,524 such questions). In this case the predictive accuracy of the agent was 96.19%.

We have analyzed in detail each case where both the developing expert and the independent expert agreed that the agent failed to recognize that a source was relevant or

Table 1: Evaluation results

Reviewer	Total number of reviewed questions	Number of IF questions with relevant sources	Number of IF questions with irrelevant sources	Time spent to review all the questions	Accuracy on IF questions with relevant sources	Accuracy on IF questions with irrelevant sources	Total accuracy
Developing expert	406	202	204	5 hours	96.53%	81.86%	89.16%
Independent expert	401	198	203	10 hours over 2 days	95.45%	76.35%	85.76%
Independent expert	1,524	198+1,326	—	22 hours for 1,326 questions	96.19%	—	—

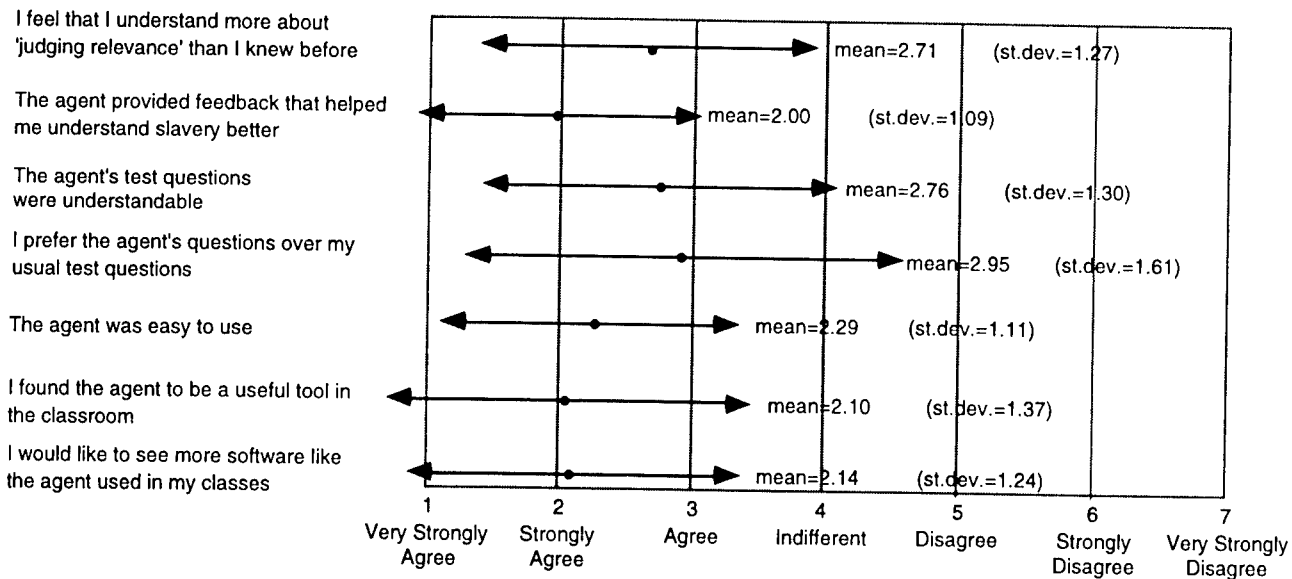


Figure 12: Student survey results

irrelevant to a certain task. In most cases it was concluded that the representation of the source was incomplete. This analysis suggested that the representation of the sources should be guided by the following principle which, if followed, would have avoided many of the agent's errors: *Any historical source must be completely described in terms of the concepts from the knowledge base.* This means that if the knowledge base contains a certain historical concept, then any historical source referring to that concept should contain the concept in the description of its content. Operationally, this simply means that if the expert decides to describe a new source in terms of some new concept C, then the expert has to review again the descriptions of each source S from the knowledge base. If the experts decides that S refers to C, then she or he has to include C in the representation of S. This does not mean, however, that the contents of the historical sources have to be completely described (a task that would be very hard, especially for pictures).

There were several cases where the two experts disagreed themselves, mainly because the independent expert had a broader interpretation of some general terms (such as slave culture, activities related to slavery, cruelty of slavery, and master slave relationships) than the developer of the knowledge base. However, the independent expert agreed that someone else could have a more restricted interpretation of those terms, and, in such a case, the answers of the agent could be considered correct. There were also 5 cases where the independent expert disagreed with the agent and then, upon further analysis of the test questions, agreed with the agent.

Table 1 indicates also the evaluation time because, unlike the automatic learning systems, the interactive learning systems require significant time from domain experts, and this factor should be taken into consideration when developing such systems. First of all, one could notice that it took twice as long to the independent expert to

analyze 401 test questions than it took to the developing expert. This is because the independent expert was not familiar with any of the 80 historical sources used in the questions, and he had to analyze each of them in detail in order to answer the questions. However, once the independent expert became familiar with the sources, he answered the new 1,326 test questions much faster.

We have also conducted an experiment with a class of 21 students from the 8th grade at The Bridges Academy in Washington D.C. The students were first given a lecture on relevance and then were asked to answer 25 test questions that were dynamically generated by the agent. Students were also asked to investigate the hints and the explanations. To record their impressions, they were asked to respond to a set of 18 survey questions with one of the following phrases: very strongly agree, strongly agree, agree, indifferent, disagree, strongly disagree, and very strongly disagree. Figure 12 presents the results from 7 of the most informative survey questions.

Finally, a user group experiment was conducted with 8 teachers at The Public School 330 in the Bronx, New York City. This group of teachers had the opportunity to review the performance of the agent and was then asked to complete a questionnaire. Several of the most informative questions and a summary of the teacher's responses are presented in Figure 13.

7 CONCLUSIONS

In this paper we have presented an innovative application of the Disciple Shell to the building of a test generation agent. We have provided experimental evidence that the process of teaching the agent is natural and efficient, and that it results in a knowledge base of good quality and in a useful educational agent. Since the agent is taught by the educator through examples and explanations, and then it is able to provide similar examples and explanations to

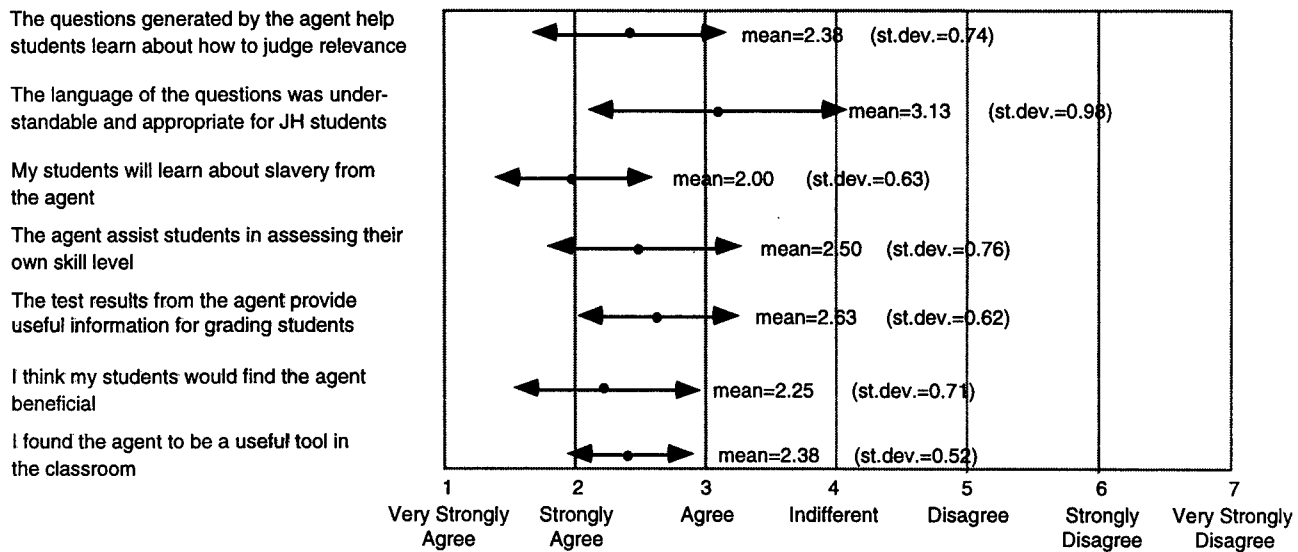


Figure 13: Teacher survey results

the students, it could be considered as being a preliminary example of a new type of educational agent that can be taught by an educator to teach the students (Hamburger and Tecuci, 1998). From the point of view of the artificial intelligence research, this work shows an integration of machine learning and knowledge acquisition with problem solving and intelligent-tutoring systems. From the point of view of the education research, it shows an automated computer-based approach to the assessment of higher-order thinking skills, as well as an assessment that involves multimedia documents. Future work involves further development of the agent and its experimental use in the classroom. We are also continuing the development of the Disciple approach and are applying it to other challenging problems, such as building a statistical analysis assessment and support agent, and an agent who has to find the best way of working around various damages to an infrastructure, such as a damaged bridge or tunnel.

Acknowledgments

Tomasz Dybala and Kathryn Wright contributed to the development of the Disciple Shell. Lynn Fontana, Rich Rosser, and David Webster assisted in the development of the knowledge base. Lawrence Young, who holds an M.S. in History Education, acted as the independent expert. This research was done in the Learning Agents Laboratory which is supported by the AFOSR grant No. F49620-97-1-0188, by the DARPA contract No. N66001-95-D-8653, and by the NSF grant No. CDA-9616478.

References

Beyer, B. (1987). *Practical Strategies for the Teaching of Thinking*. Allyn and Bacon, Inc. Boston, MA.
 Beyer, B. (1988). *Developing a Thinking Skills Program*. Allyn and Bacon, Inc. Boston, MA.
 Bloom, B. (1956). *Taxonomy of Educational Objectives*. David McKay Co., Inc. New York.

DeJong, G. and Mooney, R. (1986). Explanation-Based Learning: An Alternative View, *Machine Learning*, Vol. 1, pp. 145-176.

Fontana, L., Debe, C., White, C. and Cates, W. (1993). Multimedia: Gateway to Higher-Order Thinking Skills in Progress. In *Proc. of the National Convention of the Assoc. for Educational Communications and Technology*.

Gruber, T.R. (1993). Toward principles for the design of ontologies used for knowledge sharing. In Guarino, N. and Poli, R. (eds), *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic.

Hamburger H. and Tecuci G. (1998). Toward a Unification of Human-Computer Learning and Tutoring, In *Proc. of ITS'98*, San Antonio, TX, Springer-Verlag.

Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, MA.

Michalski, R.S. and Tecuci, G., (editors), (1994) *Machine Learning: A Multistrategy Approach, Volume 4*, Morgan Kaufmann Publishers, San Mateo, CA.

Mitchell, T.M., Keller, T., and Kedar-Cabelli, S. (1986) Explanation-Based Generalization: A Unifying View, *Machine Learning*, Vol. 1, pp. 47-80.

Mitchell T.M., Mahadevan S., and Steinberg L.I. (1985). LEAP: A Learning Apprentice System for VLSI Design, in *Proceedings of IJCAI-85*.

Quillian, M. R. (1968). Semantic Memory, In Minsky, M. (editor), *Semantic Information Processing*, pp. 227-270, Cambridge, Mass: MIT Press.

Tecuci, G. and Kodratoff, Y. (editors), (1995). *Machine Learning and Knowledge Acquisition: Integrated Approaches*, Academic Press.

Tecuci G. (1998). *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*, Academic Press.

The Problem with Noise and Small Disjuncts

Gary M. Weiss* and Haym Hirsh
 Department of Computer Science
 Rutgers University
 New Brunswick, NJ 08903
 gmweiss@att.com, hirsh@cs.rutgers.edu

Abstract

Many systems that learn from examples express the learned concept as a disjunction. Those disjuncts that cover only a few examples are referred to as small disjuncts. The problem with small disjuncts is that they have a much higher error rate than large disjuncts but are necessary to achieve a high level of predictive accuracy. This paper investigates the effect of noise on small disjuncts. In particular, we show that when noise is added to two real-world domains, a significant, and disproportionate number of the total errors are contributed by the small disjuncts; thus, in the presence of noise, it is the small disjuncts that are primarily responsible for the poor predictive accuracy of the learned concept.

1 INTRODUCTION

Systems that learn from examples often express the learned concept as a disjunction. The coverage, or size, of each disjunct is defined as the number of training examples that it correctly classifies (Holte, Åcker & Porter, 1989). Small disjuncts are those disjuncts that cover only a few training examples. Although small disjuncts may individually cover only a small fraction of the training examples, collectively they can cover a significant percentage of the training examples. The *problem with small disjuncts* is that they have a higher error rate than large disjuncts but cannot be eliminated without greatly reducing the predictive accuracy of the learned concept.

Early work on small disjuncts investigated a variety of issues, including ways of improving predictive accuracy by eliminating some small disjuncts (Holte, et al., 1989; Quinlan, 1991). Danyluk and Provost (1993) highlighted the role of small disjuncts in learning from noisy data when they speculated that in the telecommunication

domain they were studying, learning from noisy data was hard due to a difficulty distinguishing between systematic noise and "true" exceptional cases in the training data. True exceptions and small disjuncts, although similar entities which are sometimes used interchangeably, differ in one important way—true exceptions are defined relative to the "true" (i.e., correct) concept whereas small disjuncts are defined relative to a learned concept. Weiss (1995) investigated the interaction of noise on true exceptions by using artificial datasets and demonstrated that this interaction results in error prone small disjuncts in the learned concept. In this paper we focus on small disjuncts rather than "true exceptions" because for the real world domains we use, the "correct" concept definition is not known, and hence it is not possible to measure the true exceptions.

This paper extends previous work by examining the effect of noise on small disjuncts using real-world datasets and assessing the impact of this effect on the overall learning process. In particular, we show that when noise is added to these datasets, then the concept learned from this data exhibits the *problem with noise and small disjuncts*; that is, the small disjuncts contribute a disproportionate, and significant, number of the total errors (relative to the number of examples they cover) but still cannot be eliminated without adversely affecting the accuracy of the learned concept. Thus, we show that the small disjuncts are primarily responsible for learning being difficult in the presence of noise.

2 DESCRIPTION OF EXPERIMENTS

This section describes the learning program, problem domains and experimental methodology we used to conduct our experiments.

2.1 THE LEARNER

All of the experiments described in this paper use C4.5, a program for inducing decision trees from preclassified training examples (Quinlan, 1993). C4.5 was chosen because it is a popular tool for learning disjunctive

*Also AT&T Labs, Middletown, NJ 07748

concepts and because we were able to modify it, without too much difficulty, to collect statistics relating to disjunct size. For the majority of experiments, C4.5 was run in one of the following two configurations:

- with its default parameters and pruning strategy, and
- with its default parameters but without any pruning and with the `-m1` option to disable the default stopping criterion.

The `-m` option stops a node from being split during the tree-building process if the resulting node covers fewer than the specified number of examples (1 in this case). Thus, in the second configuration, C4.5 will build a decision tree that correctly classifies all training examples if the examples are consistent.

2.2 THE PROBLEM DOMAINS

This paper uses the KPa7KR chess endgame (Shapiro, 1987) and Wisconsin breast cancer (Wolberg, 1990) datasets, which were obtained from the UCI repository of machine learning databases (Murz & Murphy, 1998). These datasets were selected because C4.5 was able to attain high levels of predictive accuracy on them; we wanted to come as close to learning the correct target concept as possible prior to the introduction of artificial noise. The KPa7KR dataset contains 3196 examples with 36 attributes, where each example represents a board position and has the class value "won" or "nowin". The Wisconsin breast cancer dataset contains 699 examples with nine attributes, with each example having the value "benign" or "malignant". The class distribution is approximately equal for the chess endgame domain and is 2:1 in favor of the benign class for the breast cancer domain. The results for the breast cancer domain closely parallel those for the chess domain and therefore in most cases we only display the results for the chess domain (all results are for the chess domain unless noted otherwise).

2.3 EXPERIMENTAL METHODOLOGY

For each experiment seven independent runs were performed and the results averaged together. For each run, 200 examples were randomly selected and placed into the training set while the remaining examples were placed into the test set. Unless stated otherwise, all measurements are based on the performance of the test set. Varying levels of randomly generated *class* noise are used in the experiments. The examples are considered initially noise-free. A noise level of $n\%$ means that with probability $n/100$ the class value is randomly selected from the *remaining* alternatives. This means that when 50% class noise is applied to a domain with two classes, there is no information-provided by the class variable.

For the experiments performed in this paper, coverage is defined in terms of the number of test examples correctly classified, since we felt that this would yield a more fair measure of the true coverage of each disjunct (just as measuring accuracy on the test set yields a more fair

measure). However, we do not believe this decision to be critical. For each graph presented in this paper, coverage is displayed on a logarithmic scale, so the behavior of the small disjuncts can be easily identified.

3 THE PROBLEM WITH SMALL DISJUNCTS

Although the focus of this paper is on the problem with noise and small disjuncts, this section will first show that the chess endgame and breast cancer domains exhibit the problem with small disjuncts. Figures 1 and 2 show the results of running C4.5 on the chess endgame and Wisconsin breast cancer domains, respectively, without any artificial noise applied to the datasets. For these figures, and for all figures in this paper with coverage on the x-axis, the value of each curve at coverage n is based on the *collective performance* of all the disjuncts with coverage less than or equal to n . Thus, the curves labeled "Examples" and "Errors" in Figures 1 and 2 show the percentage of total examples and errors, respectively, covered by these disjuncts (i.e., with size $\leq n$) when the learned concept is applied to the test set. The error rate curve shows the error rate of the disjuncts with size $\leq n$.

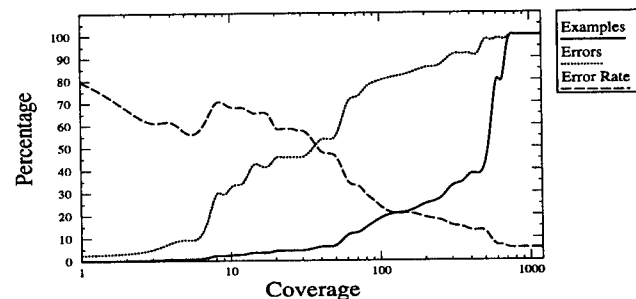


Figure 1: The Effect of Disjunct Size (Chess Domain)

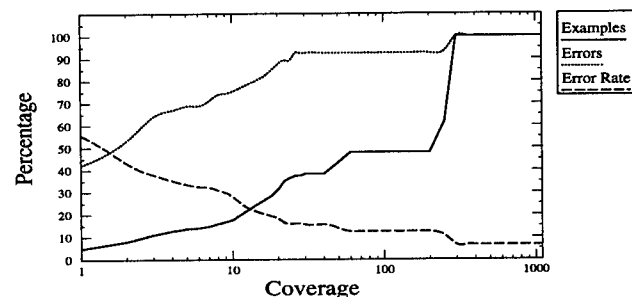


Figure 2: The Effect of Disjunct Size (Cancer Domain)

An example will help clarify the meanings of these curves and demonstrate that small disjuncts are "error prone". In Figure 1, the curves for errors and error rate intersect at coverage 40. The curves tell us that the disjuncts with size ≤ 40 collectively have an error rate of 50% and collectively cover 50% of the total errors, but only cover 5% of the total examples. This clearly demonstrates that small disjuncts are error prone (i.e., they cover a disproportionate number of errors). The error rate for the learner as a whole can be found by

looking at the error rate when 100% of the errors and examples have been covered; we see from this that the overall error rate for the chess endgame domain is 5% and the overall error rate for the breast cancer domain is 6%. The error rate curve also shows that small disjuncts have a higher error rate than large disjuncts, since the error rate decreases (for both domains) as larger disjuncts are included in the error rate calculations.

Figures 1 and 2 show that most examples are covered by the larger disjuncts, but the smaller disjuncts nonetheless cover a large percentage of the examples. This is more evident for the breast cancer domain, but even for the chess endgame domain disjuncts of size ≤ 100 are much more error prone than the larger disjuncts and cover about 20% of the total examples. These results are consistent with those described by Holte and colleagues (1989). In addition, since the small disjuncts cover too many examples to be simply dropped from the learned concept without significantly impacting the accuracy of the concept, these results also demonstrate that these domains exhibit the problem with small disjuncts.

4 THE PROBLEM WITH NOISE AND SMALL DISJUNCTS

This section will show that for the chess and breast cancer domains, noise results in small disjuncts being mainly responsible for the errors in the learned concept. For these experiments, no pruning is done unless specified and class noise is applied to both the training and test sets.

Figure 3 shows what happens to the error rate as the noise rate is varied (recall that for coverage of n , the "collective" error rate is based on all disjuncts with size $\leq n$). The figure shows that the addition of 5% class noise causes the error rate for small disjuncts to increase, but from that point on it decreases as more noise is added.

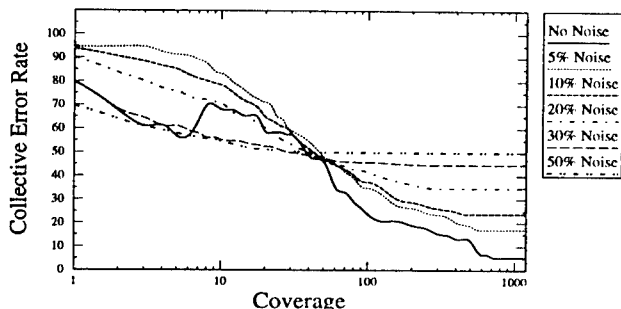


Figure 3: Effect of Noise on Error Rate

To make it easier to see the degree to which errors are concentrated toward the small disjuncts, we will use a statistic called the error factor, first introduced by Weiss (1995). The error factor is defined as:

$$\text{Error Factor}(\text{cov}) \equiv \frac{\% \text{ cumulative errors}(\text{cov})}{\% \text{ cumulative examples}(\text{cov})}$$

The error factor is a function of coverage and is essentially the "Errors" curve divided by the "Examples" curve. For example, the error factor at coverage 40 in Figure 1 is 10 (50%/5%), which indicates that disjuncts with size ≤ 40 contribute 10 times more errors than expected if coverage had no effect on error rate.

Figure 4, which plots the error factor versus coverage, shows the effect of noise on small disjuncts even more clearly than Figure 3, since error factor is a relative measure which takes into account the different overall error rates resulting from learning with the different levels of class noise. Figure 4 shows that as the amount of noise increases the error factor for small disjuncts decreases. This indicates that as the noise level increases either the percentage of errors contributed by the small disjuncts decreases and/or the percentage of examples covered by the small disjuncts increases.

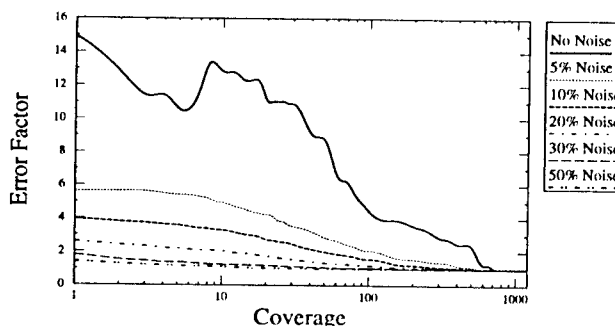


Figure 4: Effect of Noise on Error Factor

Noise added to the training data will undoubtedly affect the concept that is learned and will therefore affect the small disjuncts in the learned concept. Figure 5 addresses this by showing how various noise levels affect the number of examples covered by the small disjuncts.

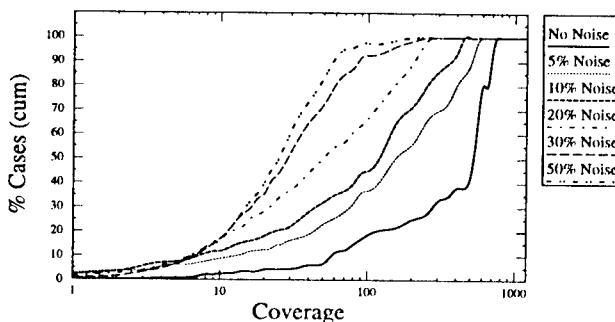


Figure 5: Effect of Noise on Distribution of Cases

Figure 5 shows that as more noise is added to the data, the number of examples covered by small disjuncts increases dramatically. For example, disjuncts of size ≤ 100 cover 3 times as many examples when the noise level increases from no noise to 10% noise. Figure 5 confirms what we and others had suspected—that noisy data will cause a learner to form "erroneous" small disjuncts.

Figure 6 shows how the distribution of errors changes as noise is applied to the domain. It shows that when the noise level is less than 20%, small disjuncts with size ≤ 30 account for an even greater percentage of the total errors than when there was no noise. Thus, we now have an explanation of why the error factor in Figure 4 decreased as additional noise was introduced—it was because the number of examples covered by the small disjuncts increased at a faster rate than the number of errors contributed by these disjuncts. Note that once the noise level reaches 30%, then disjuncts with coverage ≤ 30 no longer cover a disproportionate number of the errors—they cover half of the errors but also cover almost half of the total examples. The breast cancer domain exhibits similar trends.

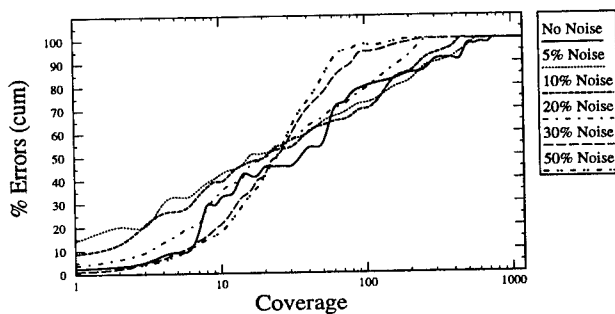


Figure 6: Effect of Noise on Distribution of Errors

We can summarize the results from Figures 3–6 as follows: in the presence of noise, small disjuncts have a higher error rate than large disjuncts and cover a significant number of the total cases and total errors. As a consequence, small disjuncts contribute a disproportionate and very significant number of the errors. All of this holds true until very high levels of noise are applied, at which point the impact of noise on the large disjuncts becomes important relative to the impact of noise on small disjuncts—at which point small disjuncts can no longer be blamed for the poor performance of the learned concept.

Since overfitting avoidance strategies such as pruning are more likely to eliminate small disjuncts than large disjuncts, it is interesting to see how these strategies will affect the error rate and how this can be related to the role of small disjuncts. Figure 7 shows how pruning affects the overall error rate. Since it is not possible to predict random class noise, the optimal error rate will equal the noise rate. This figure shows that the default pruning strategy improves the error rate in the presence of class noise and improves it the most when the noise rate is between 10% and 20%. This is explained by the fact that in this range the small disjuncts have very high error rates (Figure 3) and contribute a very large percentage of the total errors (Figure 6). The strategy which uses C4.5's -m20 option to prevent nodes from being formed when fewer than 20 examples are covered also improves the error rate, except when there is no noise. This strategy also outperforms the default pruning strategy when there are very high levels of noise (e.g., 30%), indicating that

in such cases a very aggressive overfitting avoidance strategy is needed to adequately learn the correct concept.

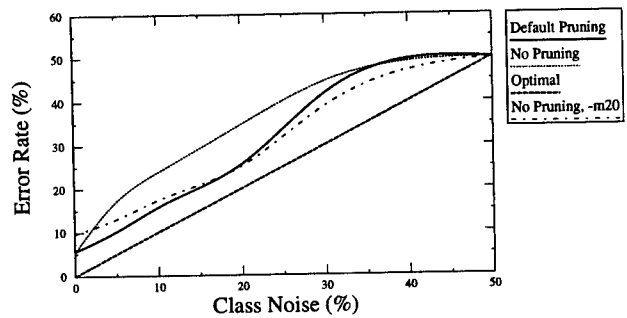


Figure 7: Effect of Pruning on Overall Error Rate

5 UNDERSTANDING THE EFFECT OF NOISE ON SMALL DISJUNCTS

In the experiments described in the previous section, the training and tests sets were generated from the same distribution. While this is the most realistic scenario, when one is trying to *understand* the effect of noise on learning, noise is frequently only applied to either the training or test set.

5.1 THE EFFECT ON TRAINING

Noise applied only to the training set tests the ability to learn the "correct" concept in the presence of noise (Quinlan, 1986). That is, by limiting the noise to the training set, we can evaluate the sensitivity of the learner to noise. We can accomplish this evaluation, even without knowing the "correct" concept, by using the noise-free test data to approximate the correct concept.

As shown earlier, noise in the training set introduces additional "erroneous" small disjuncts into the learned concept. Experiments identical to those described earlier were repeated with the artificial noise restricted to the training set. Graphs corresponding to those shown in Figures 3–6 were generated. The results indicated that under these circumstances small disjuncts have an even more significant impact on learning and, in particular, contribute a greater percentage of the errors than when noise was applied to both the training and test sets.

5.2 THE EFFECT ON TESTING

It is also meaningful to study the effect of noise on the test set. This situation corresponds to the scenario in which the training data is "cleaned up", perhaps by using more costly measurement equipment, in the hope of achieving improved predictive accuracy.¹ Experiments in which the noise was limited to the test set were run and the results showed that, relative to the case where noise

¹ However, if systematic noise is applied to the test set, better predictive accuracy may be obtained by leaving the noise in the training set.

was applied to both the training and test sets, the small disjuncts had much less of a negative impact on learning.

5.3 DISCUSSION

The results described in the previous two subsections can be explained by examining how noise affects small disjuncts. First of all, noise in the training set will influence the concept that is learned but noise in the test set cannot. Since small disjuncts are based on the learned concept, we can conclude that noise in the test set cannot cause small disjuncts to be formed. Furthermore, noise in the test set will tend to affect all disjuncts equally (Weiss, 1995). This explains why the effect of noise on small disjuncts is less dramatic when noise is applied to both the training and test sets than when it is limited to the training set—in the former case noise in the test set reduces the relative difference in error rates between the small and large disjuncts. When noise is applied to only the test set, the effect is greatly diminished, and would disappear completely if the learner were able to learn the correct concept prior to the introduction of artificial noise. For a more in depth description about *how* noise affects small disjuncts, refer to Weiss (1995).

6 CONCLUSION

This paper investigated the effect of noise on small disjuncts and how this effect impacts the overall learning process. For both the KP7KR chess end-game domain and the Wisconsin breast cancer domain, the experimental results in this paper show that small disjuncts are responsible for learning being difficult. Only at very high levels of class noise do the large disjuncts contribute a relatively large percentage of the total errors. This paper also showed some trends and effects that we feel are likely to hold for learning in general and not just for the two domains used in this paper. In particular, we feel that 1) noise tends to decrease the number of large disjuncts and increase the number of small disjuncts in the learned concept, 2) relatively low levels of noise will increase the percentage of errors contributed by small disjuncts, but this effect will diminish as higher levels of noise are applied, and 3) noise in the test set has an equalizing effect which decreases the impact of the small disjuncts on learning.

We believe these results are important because they provide some insight into *how* noise affects learning and how the effect of noise manifests itself in the learned concept. Given the prevalence of noise in real-world problem domains, such an understanding is critical. This work also provides additional justification for overfitting avoidance strategies and hopefully provides some additional insights into why these strategies work, how they can be improved and the limitations of such strategies.

Acknowledgements

Thanks to Andrea Danyluk, Foster Provost and Rob Holte for helpful comments and interesting discussions on the role of small disjuncts in learning. The authors would also like to thank the members of the Rutgers Machine Learning Research Group for their many constructive comments.

References

- Danyluk, A. P. & Provost, F.J. (1993). Small disjuncts in action: learning to diagnose errors in the local loop of the telephone network. In *Machine Learning: Proceedings of the Tenth International Conference*, 81-88, San Francisco, CA: Morgan Kaufmann.
- Holte, R. C., Acker, L. E., & Porter, B. W. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 813-818. San Mateo, CA: Morgan Kaufmann.
- Merz, C. J., & Murphy, P. M. (1998). *The UCI Repository of machine learning databases* [<http://www.ics.uci.edu/mllearn/MLRepository.html>]. Irvine, CA: University of California, Department of Information and Computer Science.
- Quinlan, J. R. (1986). The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (eds.), *Machine Learning, an Artificial Intelligence Approach, Volume II*, 149-166, Morgan Kaufmann.
- Quinlan, J. R. (1991). Technical note: improved estimates for the accuracy of small disjuncts. *Machine Learning*, 6(1), 93-98.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- Shapiro, A. D. (1987). *Structured Induction in Expert Systems*, Addison-Wesley.
- Weiss, G. M. (1995). Learning with rare cases and small disjuncts, In *Machine Learning: Proceedings of the Twelfth International Conference*, 558-565, San Francisco, CA: Morgan Kaufmann.
- Wolberg, W. H. & Mangasarian, O.L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences, U.S.A.* (Vol 87), 9193-9196.

Index

Abe, Naoki, 1, 395
Aler, Ricardo, 10
Alstrøm, Preben, 463
Anglano, Cosimo, 19
Atighetchi, Michael, 430

Baxter, Jonathan, 28
Bay, Stephen D., 37
Bello, Giuseppe Lo, 19
Billsus, Daniel, 46
Blockeel, Hendrick, 55, 136
Bollacker, Kurt D., 64
Bonet, Blai, 73
Borrajó, Daniel, 10
Boyan, Justin A., 386, 522
Bradley, Elizabeth, 547
Bradley, Paul S., 82, 91

Campbell, Colin, 188
Carberry, Sandra, 497
Cesa-Bianchi, Nicolò, 100
Cohen, Paul R., 430
Cristianini, Nello, 109, 188

De Raedt, Luc, 55, 136
Dietterich, Thomas G., 118
Domingos, Pedro, 127
Džeroski, Sašo, 136

Fawcett, Tom, 445
Fayyad, Usama M., 91
Fischer, Paul, 100
Frank, Eibe, 144, 152
Fratkina, Raya, 287
Freitag, Dayne, 161
Freund, Yoav, 170
Frieß, Thilo-Thomas, 188
Friedman, Nir, 179

Gábor, Zoltán, 197
Gama, João, 206
Gammerman, A., 515
Garcia, Frédérick, 215

Geffner, Héctor, 73
Ghosh, Joydeep, 64
Giordana, Attilio, 19
Goldszmidt, Moises, 179
Gordon, Diana F., 224

Heskes, Tom, 233
Hirsh, Haym, 574
Hu, Junling, 242

Isasi, Pedro, 10
Iyer, Raj, 170

Jappy, Pascal, 413
Jiang, Fan, 314
Juillé, Hugues, 251

Kaelbling, Leslie P., 531
Kalmár, Zsolt, 197
Kearns, Michael, 260, 269
Keeling, Harry, 565
Keim, Greg A., 314
Kimura, Hajime, 278
Kobayashi, Shigenobu, 278
Kohavi, Ron, 445
Koller, Daphne, 287
Kosko, Bart, 377

Lee, Mary S., 386
Lee, Thomas J., 179
Lin, Dekang, 296
Liquiere, Michael, 305
Littman, Michael L., 314
Loch, John, 323

Mamitsuka, Hiroshi, 1
Mangasarian, Olvi L., 82
Mansour, Yishay, 269
Margaritis, Dimitris, 332
Maron, Oded, 341
McCallum, Andrew K., 350, 359
McCluskey, T. L., 368
McGarity, Michael J., 421

Mitaim, Sanya, 377
Mitchell, Tom, 359
Mooney, Raymond J., 454
Moore, Andrew W., 386, 522

Nakamura, Atsuyoshi, 395
Ndiaye, Seydina M., 215
Ng, Andrew Y., 359, 404
Nigam, Kamal, 350
Nock, Richard, 413

Pazzani, Michael J., 46
Pendrith, Mark D., 421, 481
Piater, Justus H., 430
Pollack, Jordan B., 251
Precup, Doina, 439, 556
Provost, Foster, 445

Ramachandran, Sowmya, 454
Ramon, Jan, 55
Randløv, Jette, 463
Ratan, Aparna Lakshmi, 341
Reddy, Chandra, 472
Rosenfeld, Ronald, 359
Ryan, Malcolm R. K., 481

Saitta, Lorenza, 19
Sallantin, Jean, 305
Sałustowicz, Rafał, 488
Samuel, Ken, 497
Saul, Lawrence K., 506
Saunders, C., 515
Schapire, Robert E., 170
Schmidhuber, Jürgen, 488
Schneider, Jeff G., 386, 522
Shatkay, Hagit, 531
Shawe-Taylor, John, 109
Singer, Yoram, 170
Singh, Satinder, 260, 323, 556
Street, W. Nick, 540
Stuart, Joshua M., 547
Sutton, Richard S., 556
Sykacek, Peter, 109
Szepesvári, Csaba, 197

Tadepalli, Prasad, 472
Tecuci, Gheorghe, 565
Thrun, Sebastian, 332
Tridgell, Andrew, 28

Utgoff, Paul E., 439

Vijay-Shanker, K., 497
Vovk, V., 515

Weaver, Lex, 28
Weiss, Gary M., 574
Wellman, Michael P., 242

West, M. M., 368
Witten, Ian H., 144, 152

Zhang, Xiaoqin, 430

Other Titles of Interest from Morgan Kaufmann Publishers

ICML 1987–1997: Proceedings of the Fourth
through the Fourteenth International Conference
on Machine Learning

Readings in Machine Learning edited by
Jude Shavlik & Thomas Dietterich

Elements of Machine Learning by Pat Langley

Readings in Agents edited by Michael N. Huhns &
Munindar P. Singh

Artificial Intelligence: A New Synthesis
by Nils J. Nilsson

Fundamentals of the Theory of Computation
by Raymond Greenlaw & James H. Hoover

Introduction to Knowledge Systems by Mark Stefik

Journal of Artificial Intelligence Research, Vols. 1–7

IJCAI 1969–1997: Proceedings of the First through
Fifteenth International Joint Conference on
Artificial Intelligence

Machine Learning Methods for Planning
edited by Steven Minton

Machine Learning: A Multistrategy Approach,
Volume IV edited by Ryszard Michalski &
Gheorghe Tecuci

C4.5: Programs for Machine Learning
by J. Ross Quinlan

Computer Systems that Learn: Classification &
Prediction Methods from Statistics, Neural Nets,
Machine Learning & Expert Systems
by Sholom M. Weiss & Casimir A. Kulikowski

Machine Learning: A Theoretical Approach
by Balas K. Natarajan

Genetic Programming: An Introduction
by Wolfgang Banzhaf, Peter Nordin, Robert E. Keller
& Frank D. Francone

Foundations of Genetic Algorithms, Volumes 1–4;
(Volume 5 forthcoming)

Forthcoming...

Genetic Programming III: Automatic Programming &
Automatic Circuit Synthesis
by John R. Koza, David Andre, Forrest H. Bennett III,
& Martin A. Keane

MORGAN KAUFMANN PUBLISHERS
340 PINE STREET, SIXTH FLOOR
SAN FRANCISCO, CA 94104
[HTTP://WWW.MKP.COM](http://www.mkp.com)

ISBN 1-55860-556-8

